MPLAB Harmony Help

MPLAB Harmony Integrated Software Framework

Volume III: MPLAB Harmony Configurator (MHC)

This volume provides user and developer-specific information on the MPLAB Harmony Configurator (MHC).

Description

The MPLAB Harmony Configurator (MHC) is a graphical utility used to configure MPLAB Harmony projects. MHC provides a "New MPLAB Harmony" project wizard and a graphical user interface for configuration of MPLAB Harmony projects. When used, it generates (or updates) a project outline, including the C-language main function and system configuration files and stores the project configuration selections for later retrieval, modification, and sharing.

Introduction

This section provides an overview of the MPLAB Harmony Configurator (MHC).

Description

The MPLAB Harmony Configurator (MHC) is a MPLAB X IDE plug-in. It must be installed into your MPLAB X IDE installation to be used. See the Installing MHC section for information on installing the MHC plug-in.

MPLAB Harmony Configurator User's Guide

This section provides user information on using the MHC.

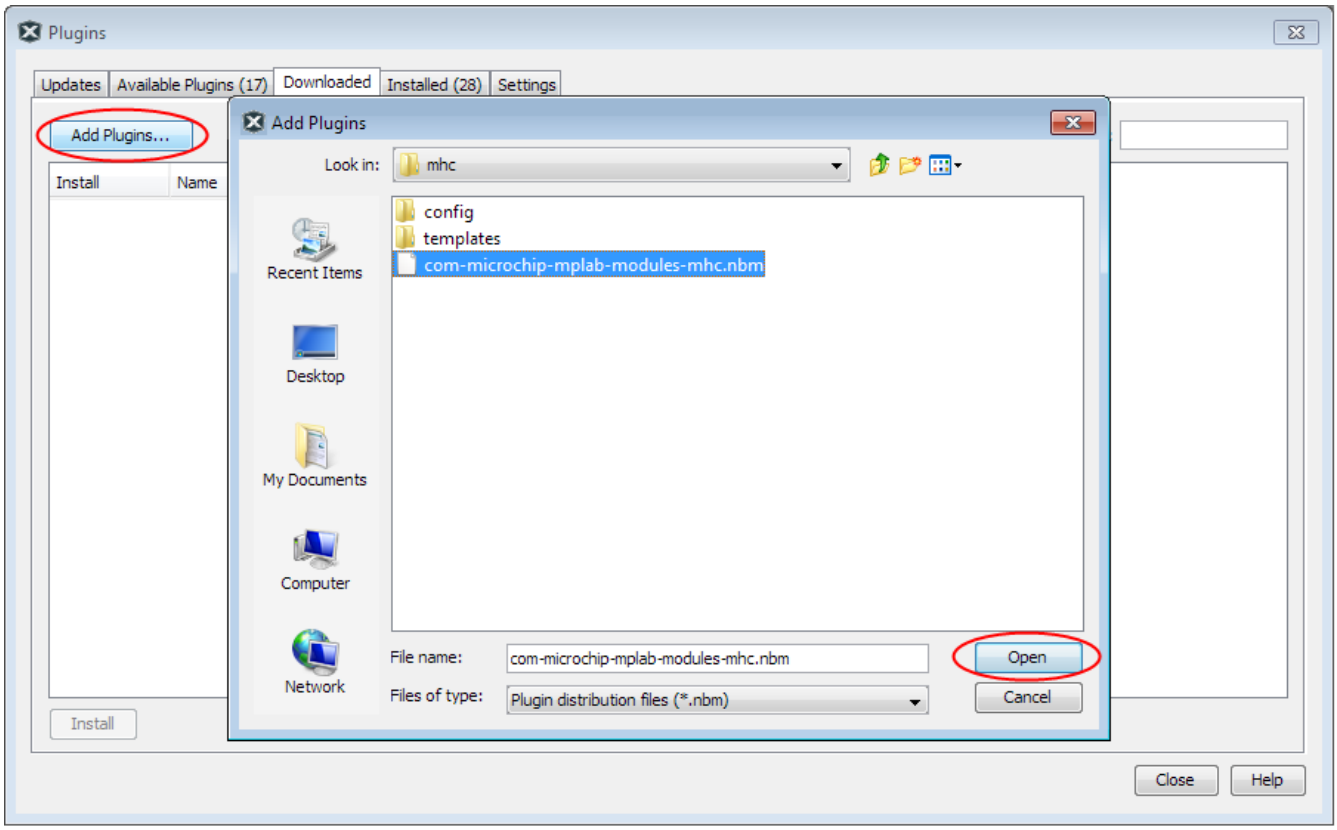
Installing MHC

This topic provides information on installing the MHC plug-in.

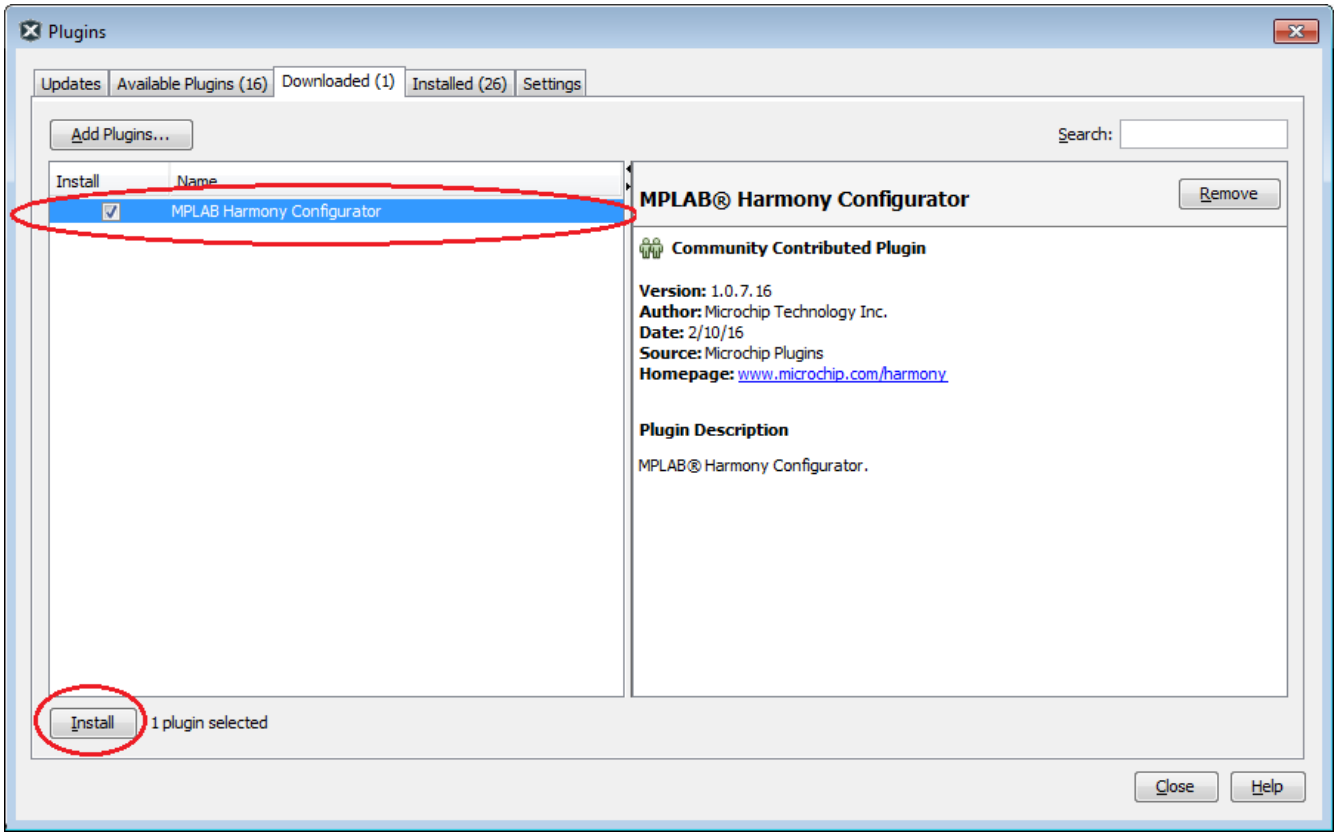
Description

Installing the MHC Plug-in

1. Start MPLAB X IDE and select *Tools > Plugins*.
2. Select the **Downloaded** tab and click **Add Plugins..**.
3. In the Add Plugins dialog, navigate to the MHC `com-microchip-mplab-modules-mhc.nbm` plug-in file, which is located in `<install-dir>/utilities/mhc`, and then click **Open**.



4. Ensure that the Install check box for the plug-in is selected and click **Install**.



5. Follow the prompts from the installation and continue until the installation completes. (Do not be concerned if the version you are installing is *signed* but not *trusted*, simply click **Continue**). Once the installation has finished you can close the **Plugins** dialog.
6. To verify the installation, select *Tools > Plugins* and select the **Installed** tab. The MHC plug-in you installed should be included in the list.

MPLAB Harmony Configurator Interface

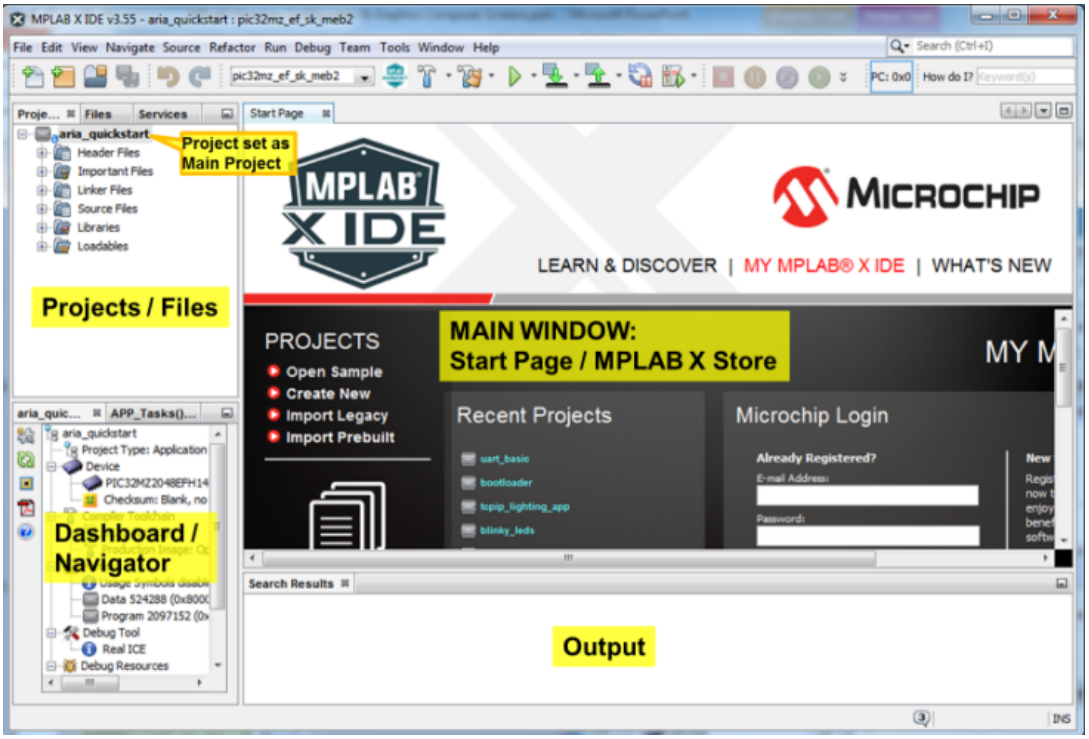
This section describes the MHC interface.

Description

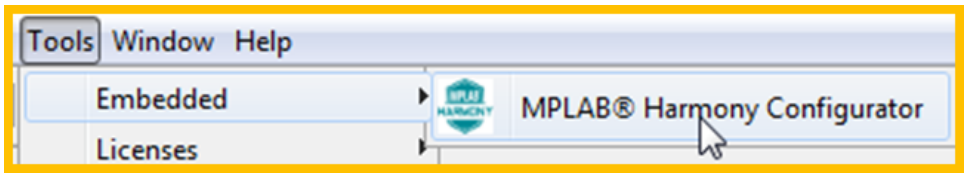
This section provides a basic overview of the MHC user interface. For detailed information on using MHC to create a MPLAB Harmony application, refer to Using MHC to Create a New Application. Most of the figures shown in this section are from screen captures of MPLAB with the Aria Quickstart project loaded. You can find this project in the MPLAB Harmony application folder `.\apps\gfx\aria_quickstart\firmware\aria_quickstart.X`. Load this project and follow along.

Quick Review of MPLAB Windows

Let's first review the windows that are visible after loading the Aria Quickstart project and setting the project as the "Main Project". The Main Project is set by right-clicking on the project name aria_quickstart and setting it as the main project. If the window configuration shown in the following figure is different than the one shown in MPLAB X IDE, select the Window:Reset Windows task from the Windows MPLAB menu. Before launching the MPLAB Harmony Configurator you should see:

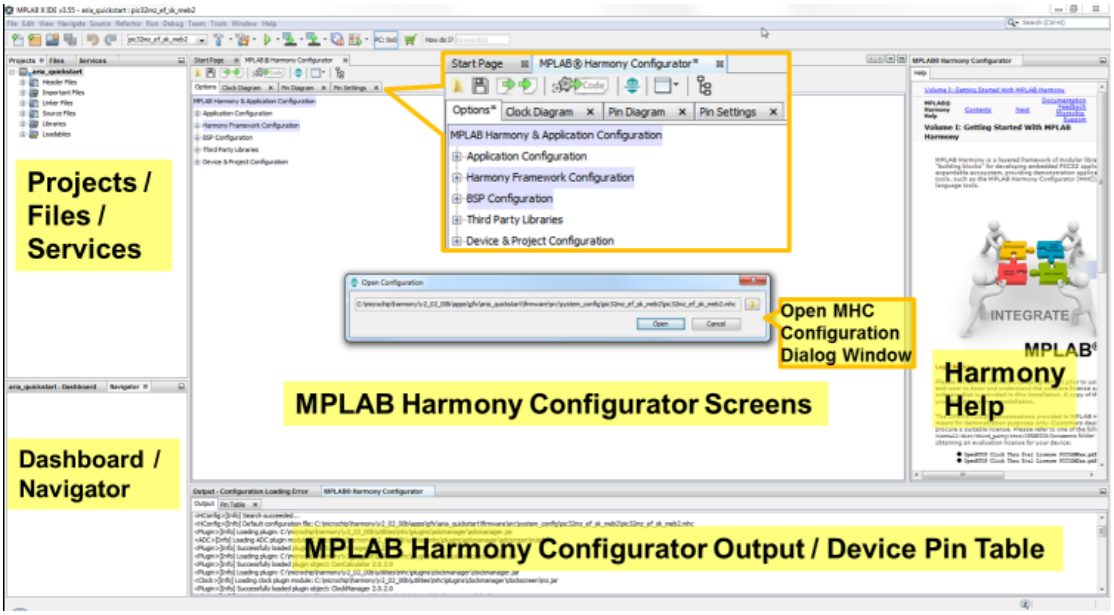


To launch the MPLAB Harmony Configurator, assuming you have already installed this plug-in, select *Tools* > *Embedded* > *MPLAB Harmony Configurator*:



Initial MHC Display

After launching the MHC, you should be prompted to load the MHC configuration (`.mhc`) file attached to the active project configuration:



The upper left window, which had Project / Files before, will update with a new third tab, Services. The upper right window appears and shows the MPLAB Harmony Configurator Help. The bottom window, which had only Output before, now has a new tab showing the Device Pin Table.

A new tab, MPLAB Harmony Configurator, has been added to the main window. There are four sub-tabs in this new window:

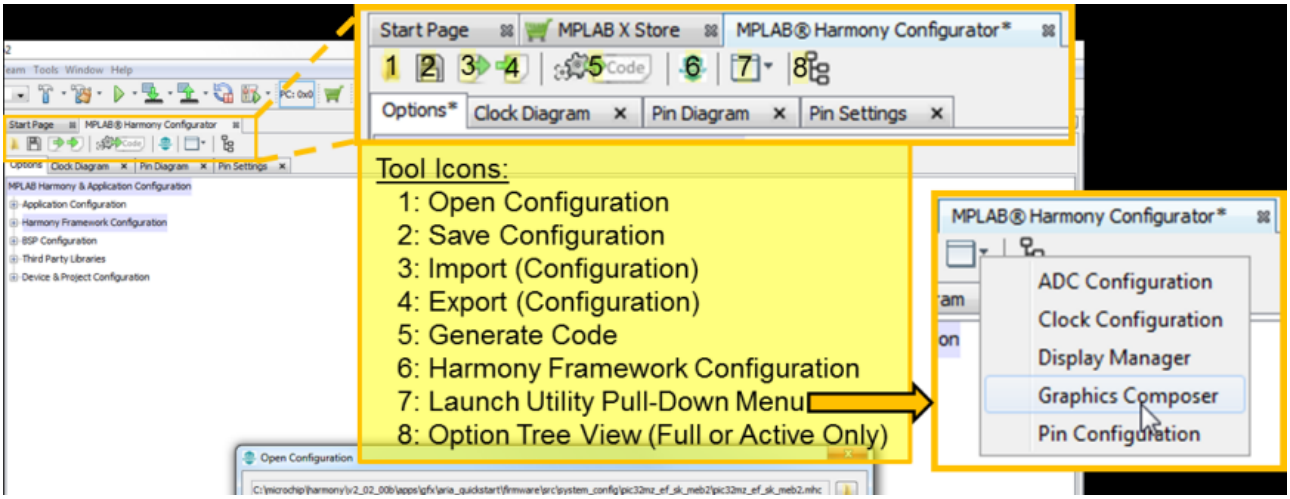
- Options – Selects various application options within MPLAB Harmony.
- Clock Diagram – Shows the current configuration of the device's clocks.
- Pin Diagram – Provides a graphical overview of how the project configures the device's pins.

- Pin Settings – Provides a tabular summary of how the device's pins are configured.

For more details on configuring pins, see *Volume III: MPLAB Harmony Configurator User's Guide* > *MPLAB Harmony Graphical Pin Manager*.

MHC Toolbar

Just below the "MPLAB Harmony Configurator" tab is a new toolbar:

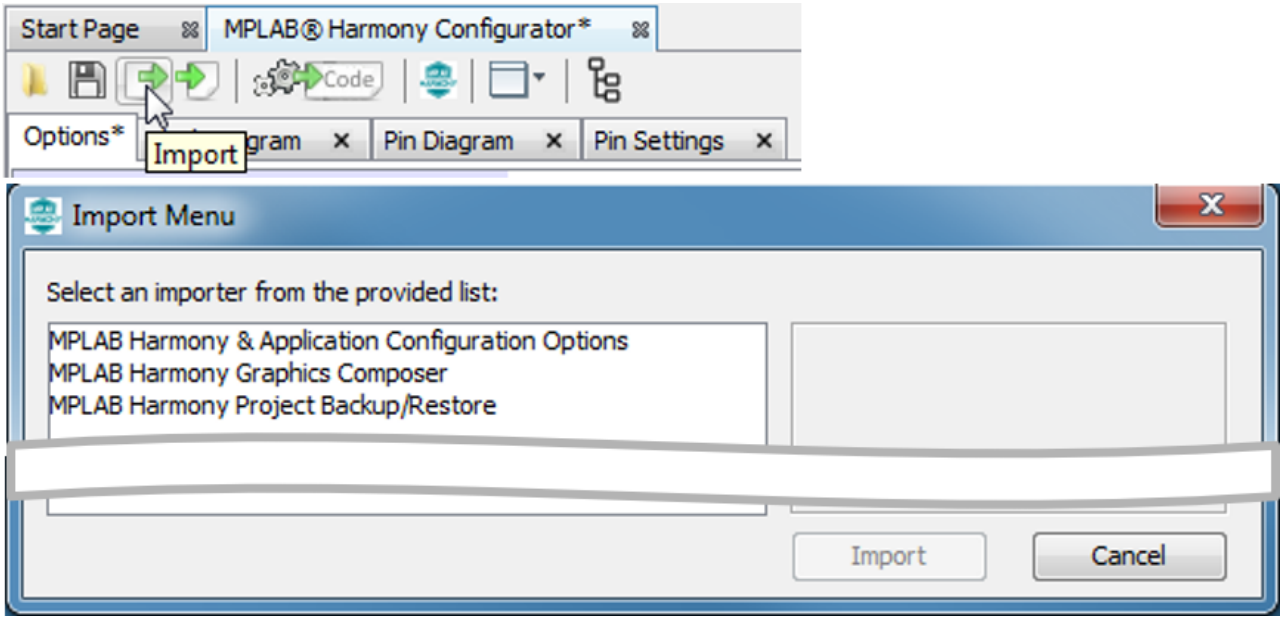


The tool icons are as follows:

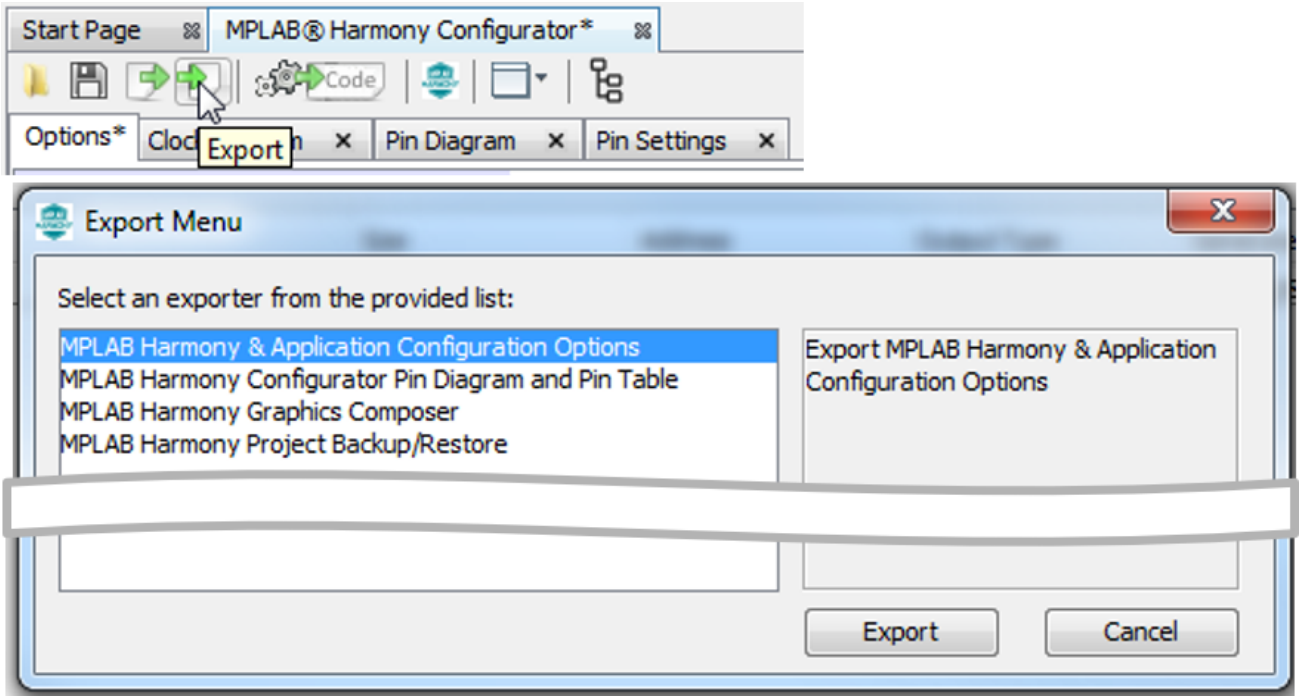
Open Configuration – Brings up a window to select and read a `.mhc` configuration file.

Save Configuration – Brings up a window to save the current MHC configuration as a `.mhc` file or to save the current Board Support Package (BSP) configuration as a new custom `.mhc` file.

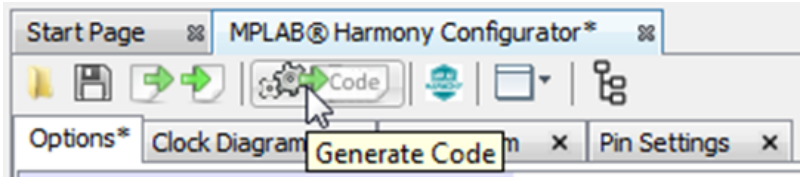
Import – This allows you to import configuration options or a project backup. The second option, MPLAB Harmony Graphics Composer, is only available when the Graphics Composer screen is active:



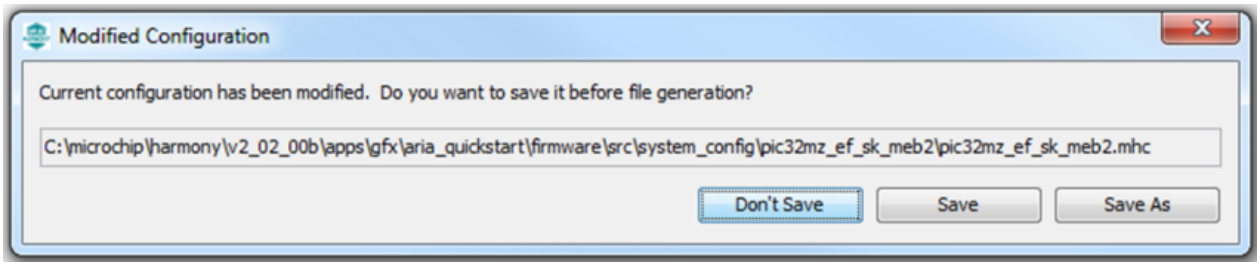
Export – Supports exporting various parts of the design. The third option, MPLAB Harmony Graphics Composer, is only available whenthe Graphics Composer screen is active:



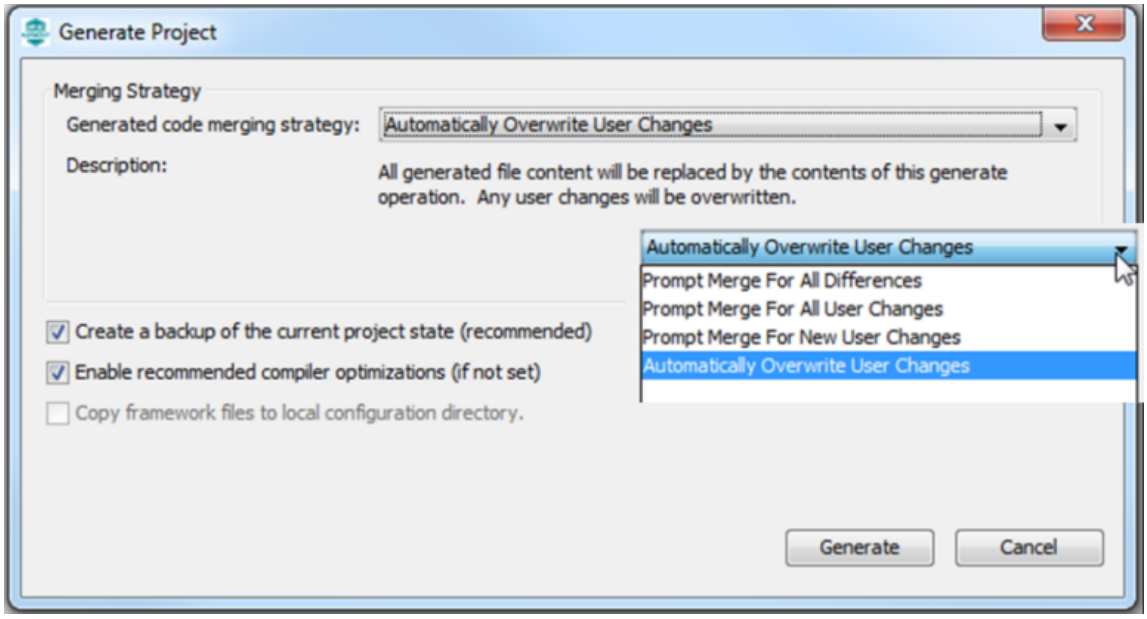
Generate Code – This initiates generating/regenerating the project's code based on the options that have been chosen. To start this process, press the Generate Code icon:



You will probably next be prompted to save the modified configuration into the project's `.mhc` file:



Finally, the Generate Project dialog will appear, allowing you to select the code merging strategy, create a backup of the project, and use the recommended compiler options for any new files created:



The available code merging strategies are:

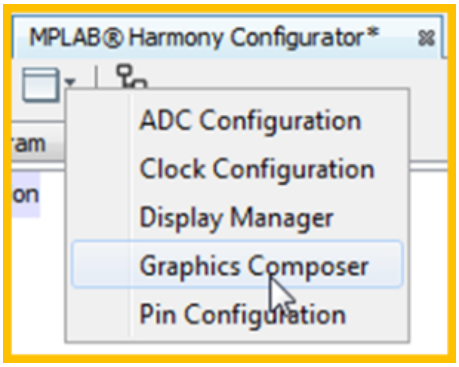
- *Prompt Merge for all Differences* – The user will be prompted with a merge window for all generated files. This includes files that have no user modifications.
- *Prompt Merge for All User Changes* – The user will always be prompted with a merge window for all generated files that contain user modifications. You should always select this merge strategy after modifying anything under the Options tab, MPLAB harmony & Application Configuration.
- *Prompt Merge For New User Changes* – The user will be prompted with a merge window for all generated files that contain user modifications. The user will not be prompted to merge changes again for a given file unless the user makes further changes to that file.
- *Automatically Overwrite User Changes* – All generated file content will be replaced by the contents of this generate operation. Any user changes will be overwritten.

Create a backup of the current project state – Provides the ability to revert all generated files to their original state, before code generation.

Enable recommended compiler optimizations (if not set) – A compiler optimization level of at least 'O1' is highly recommended for MPLAB Harmony projects. This option will set the compiler optimization level to 'O1' if no optimization level is currently set.

Copy framework files to local configuration directory – Provides the ability to generate standalone project. All necessary files will be added into MPLAB X IDE project, so it can be built and run without MPLAB Harmony framework.

Launch Utility Pull-Down Menu – Supports launching these applications:



For more information on these options:

- ADC Configuration: *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony ADC Manager User's Guide*
- Clock Configuration: *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Configurator User's Guide > Configuring the Oscillator Module Using the MHC Clock Configurator*
- Display Manager: *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Display Manager User's Guide*
- Graphics Composer: *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide*
- Pin Configuration: *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Configurator User's Guide > MPLAB Harmony Graphical Pin Manager*

Option Tree View – Selecting this icon toggles the option tree between "Global" and "Active" views.

Using MHC to Create a New Application

Provides information on creating a new MHC project.

Introduction

This section provides an introduction to creating your own MPLAB Harmony applications using the MPLAB Harmony Configurator (MHC).

Description

MPLAB Harmony provides a MPLAB Harmony Configurator (MHC) MPLAB X IDE plug-in that can be installed in MPLAB X IDE to help you create your own MPLAB Harmony applications.

To create a new MPLAB Harmony application with MHC, follow these three steps:

- Step 1: Create the New Harmony Project
- Step 2: Add and Configure Required Libraries/Modules
- Step 3: MPLAB Harmony Application Structure and Developing the Application

Note: If you are a Microchip Libraries for Applications (MLA) user, and will be porting your application from the MLA TCP/IP, File System, USB Device, Graphics, or peripheral libraries to the MPLAB Harmony equivalents, refer to Porting to MPLAB Harmony for more information.

Prerequisites

This topic describes the prerequisites for creating your own MPLAB Harmony applications using MHC.

Description

This tutorial assumes that you have already completed these steps before you start:

1. Installed the MPLAB X IDE (http://www.microchip.com/mplabx).
2. Installed MPLAB Harmony (http://www.microchip.com/harmony).
3. Installed the MPLAB XC32 C/C++ Compiler (http://www.microchip.com/xc32).
4. Set up a working PIC32 development platform (http://www.microchip.com/32bit).

You can download the MPLAB X IDE, MPLAB Harmony and the MPLAB XC32 C/C++ Compiler from the links provided. If you do not already have a PIC32 development platform, you can learn more about the PIC32 family and determine which hardware platform best meets your development needs by visiting the 32-bit website listed previously.

This tutorial also assumes that you have some familiarity with the MPLAB X IDE, embedded C-language programming and PIC32 microcontrollers. If you are unsure how to complete some of the steps in this tutorial, please refer to the documentation for the item on which you have questions. You may also seek assistance from your peers on the Microchip discussion forums (http://www.microchip.com/forums) or from the Microchip support staff (www.microchip.com/support).

Once you have everything installed, connected, and up and running you are ready to begin creating your own MPLAB Harmony applications.

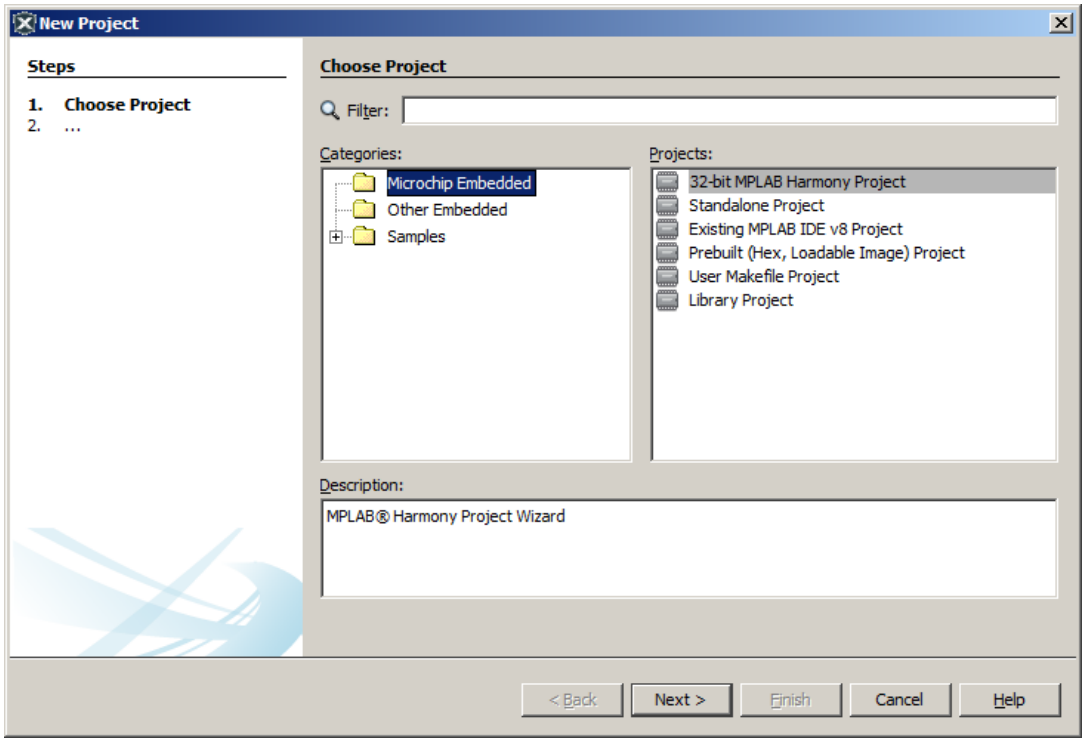
Step 1: Create the New Project

To create a new MPLAB Harmony project, you first need to create a new MPLAB X IDE project and the basic set of source code files and functions that are necessary for a properly formed MPLAB Harmony application.

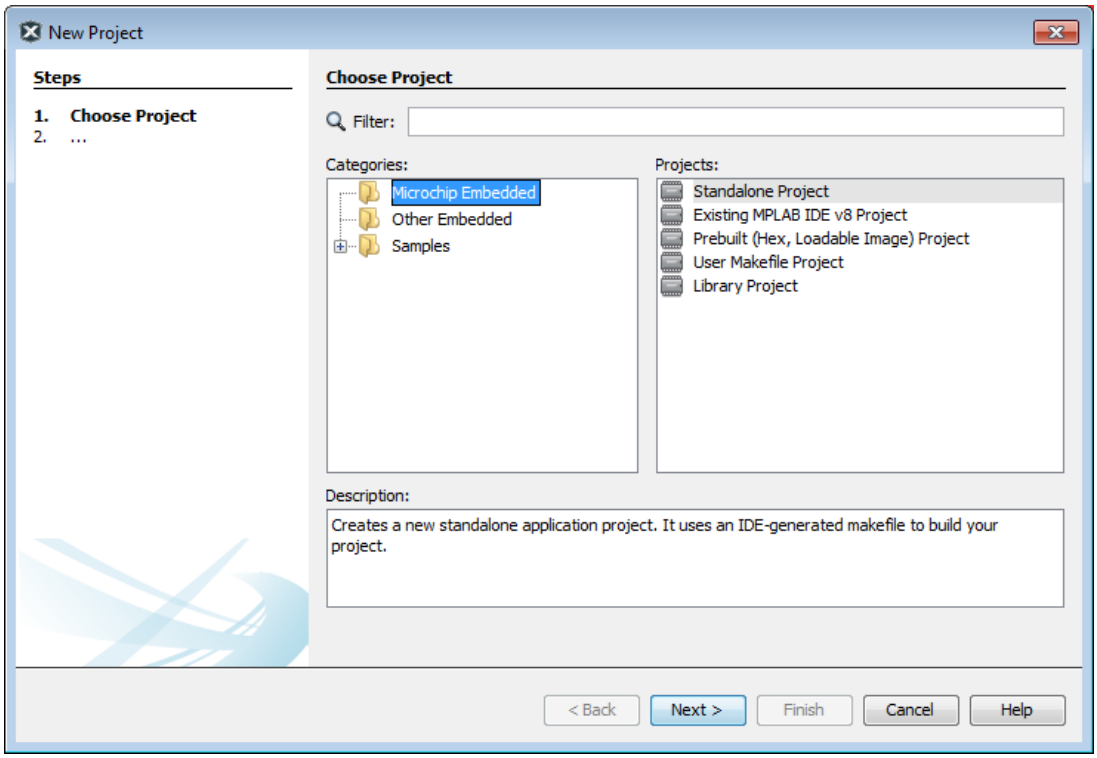
Description

To create a new MHC project:

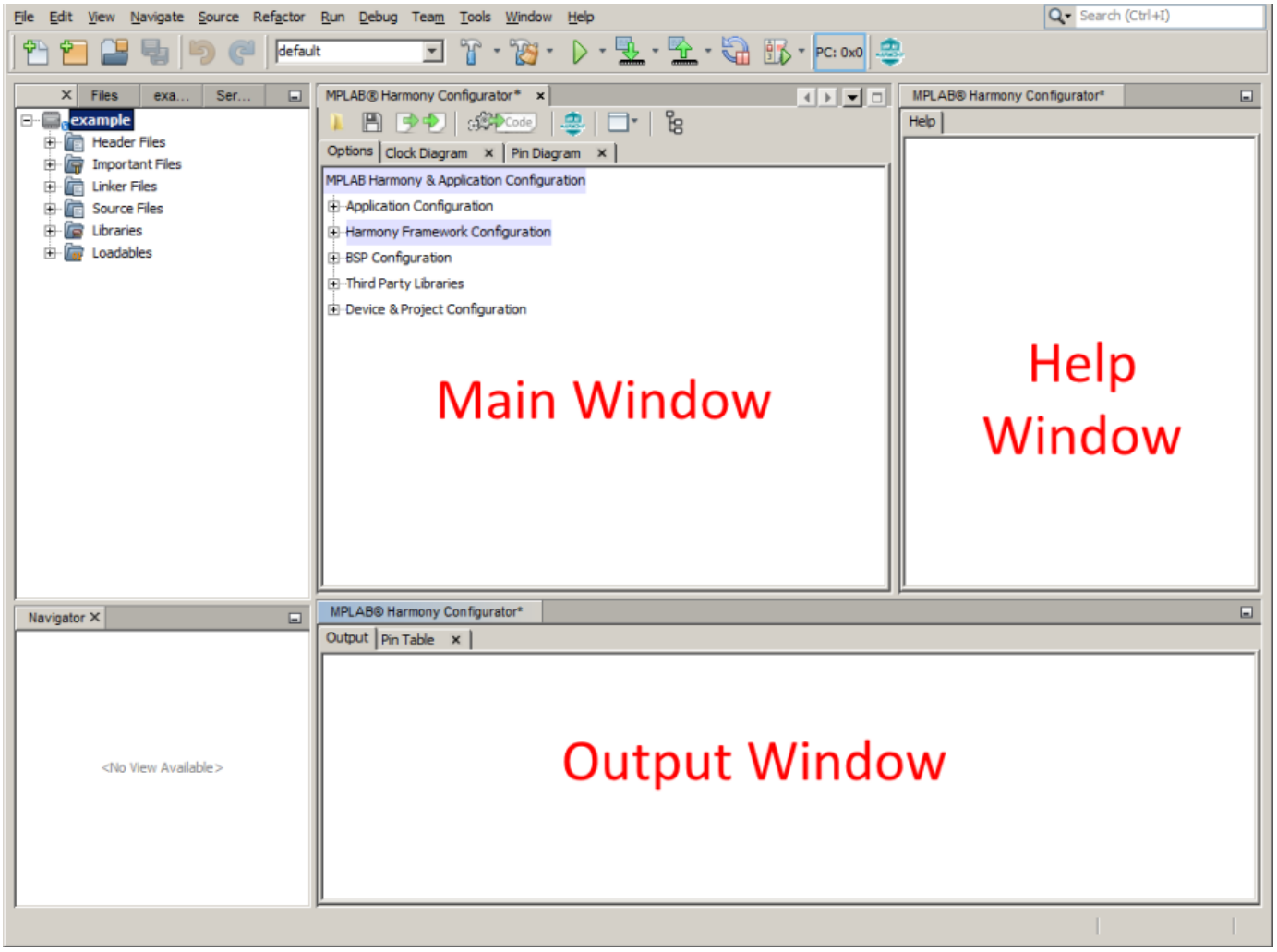
1. Select *File* > *New Project* or click the New Project icon in MPLAB X IDE.
2. In Categories, select **Microchip Embedded** and in Projects select **MPLAB Harmony Project** from the list of available project templates, and then click **Next** to launch the Microchip Harmony Configurator Project Wizard.



3. Specify the following in the New Project dialog:
   - Harmony Path (path to the folder containing Harmony framework: `<install-dir>`)
   - Project Location (the default project path is the `apps` folder within the selected MPLAB Harmony path)
   - Project Name
   - Configuration Name (optional)
   - Target Device (when a valid harmony path is selected, the device selection menu will be filled)



4. A MPLAB Harmony project will be created and the MPLAB Harmony Configurator will open. Refer to MPLAB Harmony Configurator for additional information.



Step 2: Add and Configure the Required Libraries and Modules

This topic describes how to configure the MPLAB Harmony library modules.

Description

1. In the Main window, expand the Device Configuration tree and select the desired device configuration settings.
2. Expand the MPLAB Harmony Project Configuration tree and select and configure the desired libraries.
3. If use of a Board Support Package is desired, expand the BSP Configuration tree and select the desired BSP.
4. When complete, generate and save the configuration.
5. Develop your application logic using the selected libraries.

At this point, you should be able to build, debug, and step through the application. Effectively, you have a running MPLAB Harmony system; however, it is not yet ready to do anything. Next, you will develop your application state machine logic and make sure the system does what you want it to do.

Step 3: MPLAB Harmony Application Structure and Developing the Application

This topic describes the steps necessary to maintain the state machines.

Description

main.c

The main.c file contains calls to the SYS_Initialize function, which initializes MPLAB Harmony modules, as well as applications. It also contains the main task execution, which calls tasks for all selected MPLAB Harmony modules, as well as the application task function, APP_Tasks.

app.c

The `app.c` file contains the APP_Initialize function that is used to place an application into its initial state. It will be called from the SYS_Initialize function. The APP_Task function, which is also contained in the `app.c` file, implements the application state machine logic. Add application code

to this task as desired.

Refer to the example applications located in the `<install-dir>/apps/` folder within your MPLAB Harmony installation for example applications for various MPLAB Harmony modules. Related documentation is available in the *Applications Help* > *Examples* section.

Porting a Legacy PLIB to MPLAB Harmony

Provides an example on how to port a legacy (i.e., prior to MPLAB Harmony) USART Peripheral Library (PLIB) demonstration application to a MPLAB Harmony application using the MPLAB Harmony Configurator (MHC).

Description

A detailed procedure for porting the legacy UART PLIB Interrupt demonstration application (`<compiler-install-dir>/examples/plib_examples/uart/uart_interrupt`) to MPLAB Harmony is provided in the Framework Help > Peripheral Library Help > Peripheral Library Porting Example .

In this example, the following assumptions are made:

- The PIC32MX795F512L device will be used; however, the process described in this section is applicable for other PIC32 devices with appropriate changes
- The Explorer 16 Development Board is the hardware used in this example
- For the v1.33 MPLAB XC32 C/C++ Compiler, the `examples` folder is not present. To view the legacy USART PLIB example, refer to v1.31 or earlier of the MPLAB XC32 C/C++ compiler.

Configuring the Oscillator Module Using the MHC Clock Configurator

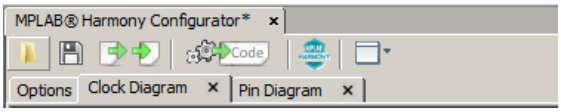
Provides information configuring the Oscillator module using the MHC Clock configurator

Description

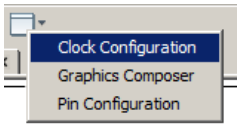
The MHC Clock Configurator is a component of the MPLAB Harmony Configurator (MHC) MPLAB X IDE plug-in. Its function is to provide a graphical user interface to configure the Oscillator module.

While simulating the normal operation of the Oscillator module, the MHC Clock Configurator contains interactive controls, dynamic output, and visual warnings to help guide the user in establishing the desired system clock configuration.

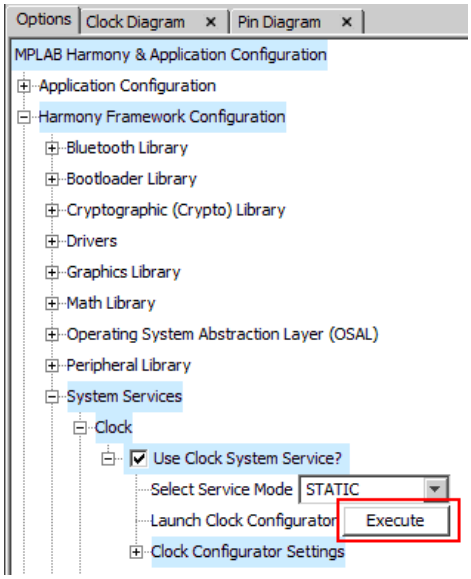
The MHC Clock Configurator is launched automatically when the MHC is launched. It is in the form of a tab panel in MPLAB X IDE. Clicking the MPLAB Harmony Clock Configuration tab will open the MHC Clock Configurator.



The clock configurator screen can also be accessed using the main window toolbar application launch feature. Simply click the application launch icon and select **Clock Configuration**.



Another way to access the MHC Clock Configurator is via the Clock System Service section in MHC Harmony & Application Configuration tree view. Pressing the Execute button at the Launch Clock Configurator topic will either bring the tab panel into focus or launch the MHC Clock Configurator, if the tab panel was closed.



Note: The MHC Clock Configurator is one option to configure the Oscillator Module. Another option is to configure directly via the MPLAB Harmony & Application Configuration tree structure. The majority of the settings captured in the MHC Clock Configurator exist under the Clock Configurator Settings node in the Clock System Service, while the remainder are in the Device Configuration section.

Clock Configuration for PIC32MZ Family Devices

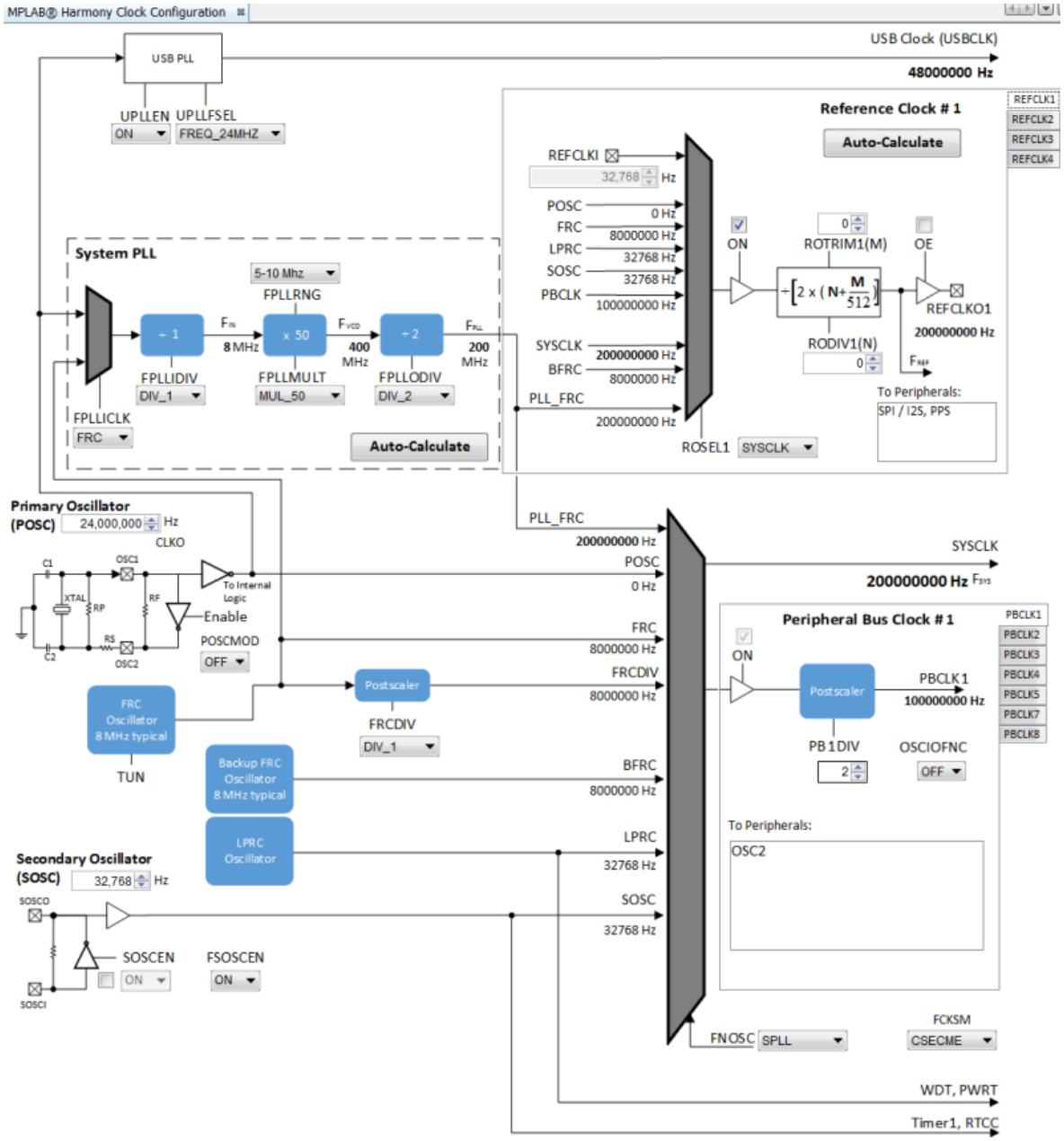
Provides configuration information for PIC32MZ family devices.

Description

The MHC Clock Configurator's support of configuring the Oscillator Module of a PIC32MZ family device is divided into the following sub-sections:

- Configuring System Clock Frequency
- Configuring the Peripheral Bus Clocks
- Configuring the Reference Clocks
- Using the SPLL Divider Auto-Calculate Feature

For details regarding the operation of the Oscillator module, refer to the **"Oscillator"** chapter in the *"PIC32MZ Embedded Connectivity (EC) Family Data Sheet"* (DS60001191). This document is available for download from the Microchip website (www.microchip.com).



Configuring the System Clock Frequency

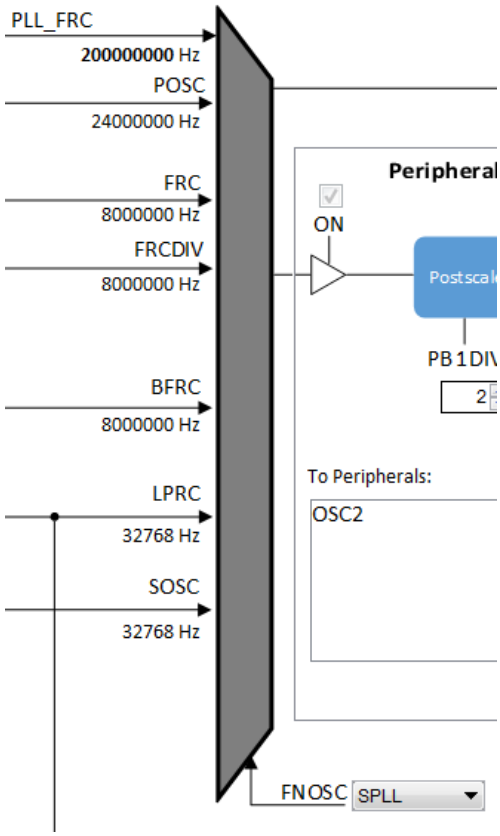
Provides information on configuring the system clock frequency for PIC32MZ family devices.

Description

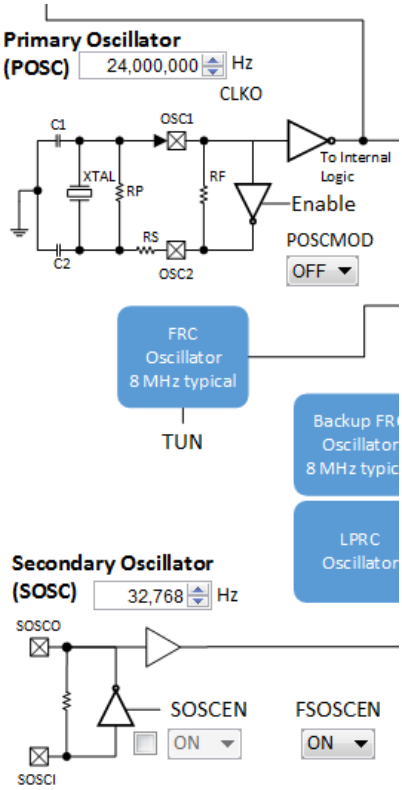
There are a total of five external and internal oscillator options as clock source:

- Internal Fast RC (FRC) Oscillator divided by the FRCDIV bits in the OSCCON register
- Internal Low-Power RC (LPRC) Oscillator
- Secondary Oscillator (SOSC)
- Primary Oscillator (POSC) (POSCMOD: HS or EC)
- System PLL (SPLL)

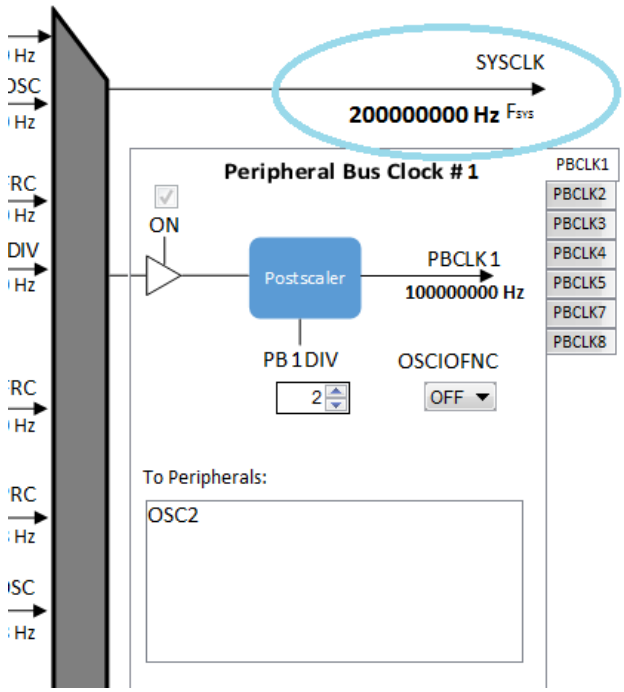
The device configuration bit FNOSC is represented as a drop-down with the above selections in the MHC Clock Configuration. The current selection is represented in **bold**.

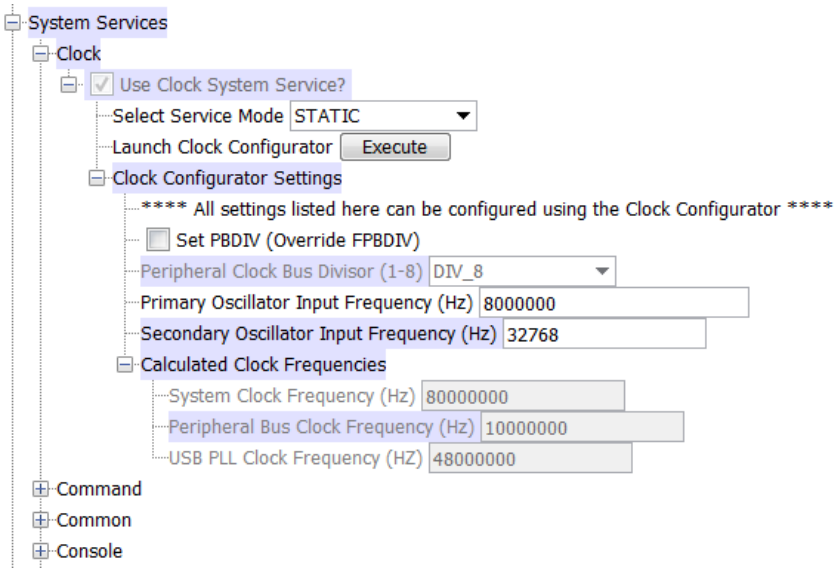


The Primary Oscillator (POSC) and Secondary Oscillator (SOSC) are customizable external clock sources. For the POSC, the device configuration bit, POSCMOD, needs to be set to EC or HS. If FNOSC is set to SOSC, the device configuration bit, FSOSCEN, should be set to ON. SOSCEN is set post-initialization. There is an option to override FSOSCEN with SOSCEN.

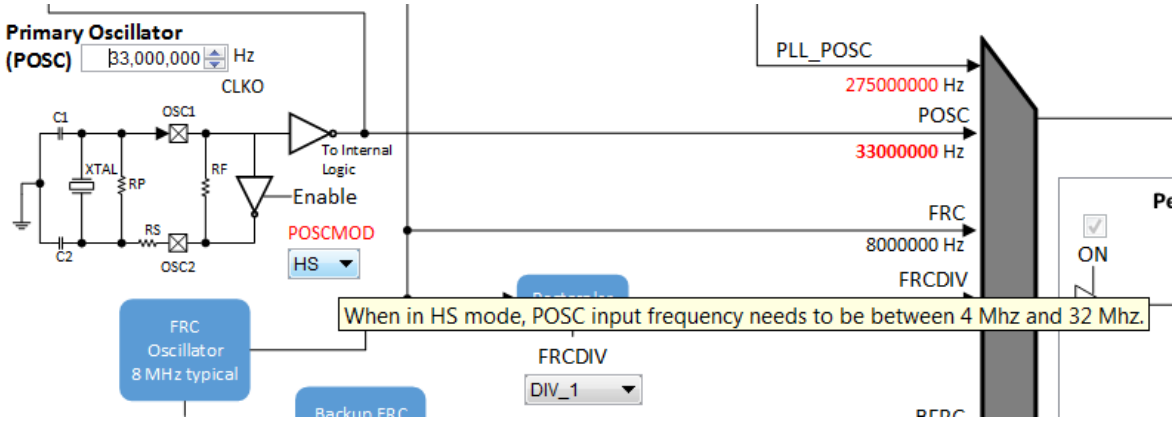


The output system clock frequency (SYSCLK) is displayed on the left side. This value (in Hz) corresponds to System Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.

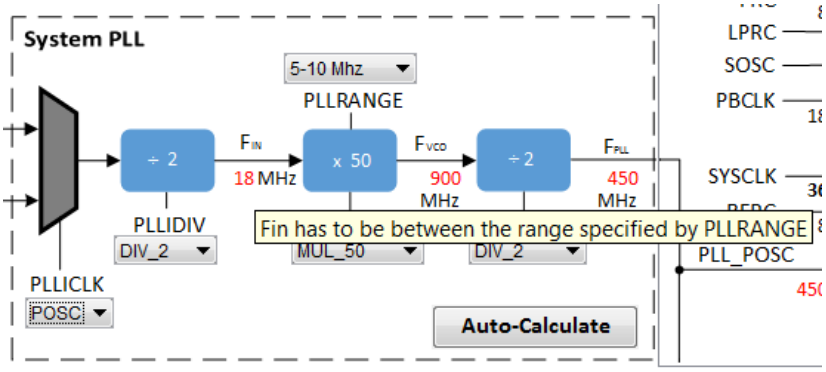




Certain frequency values may be displayed in red when the input value does not meet specification and may cripple performance of the device. An example is shown in the following figure, when the HS Oscillator Mode is selected for POSCMOD and the POSC input frequency set is outside of the 4 MHz - 32 MHz range. A dynamic help tip will also appear if the user hovers over the POSCMOD control or any of the red text.



Another example is the SPLL, where FPLL (60 MHz – 120 MHz), FVCO (80 MHz – 240 MHz), and FIN (range specified by PLLRANGE) will appear as red text, including an explanation tool tip, if they fall outside of their respective required ranges.

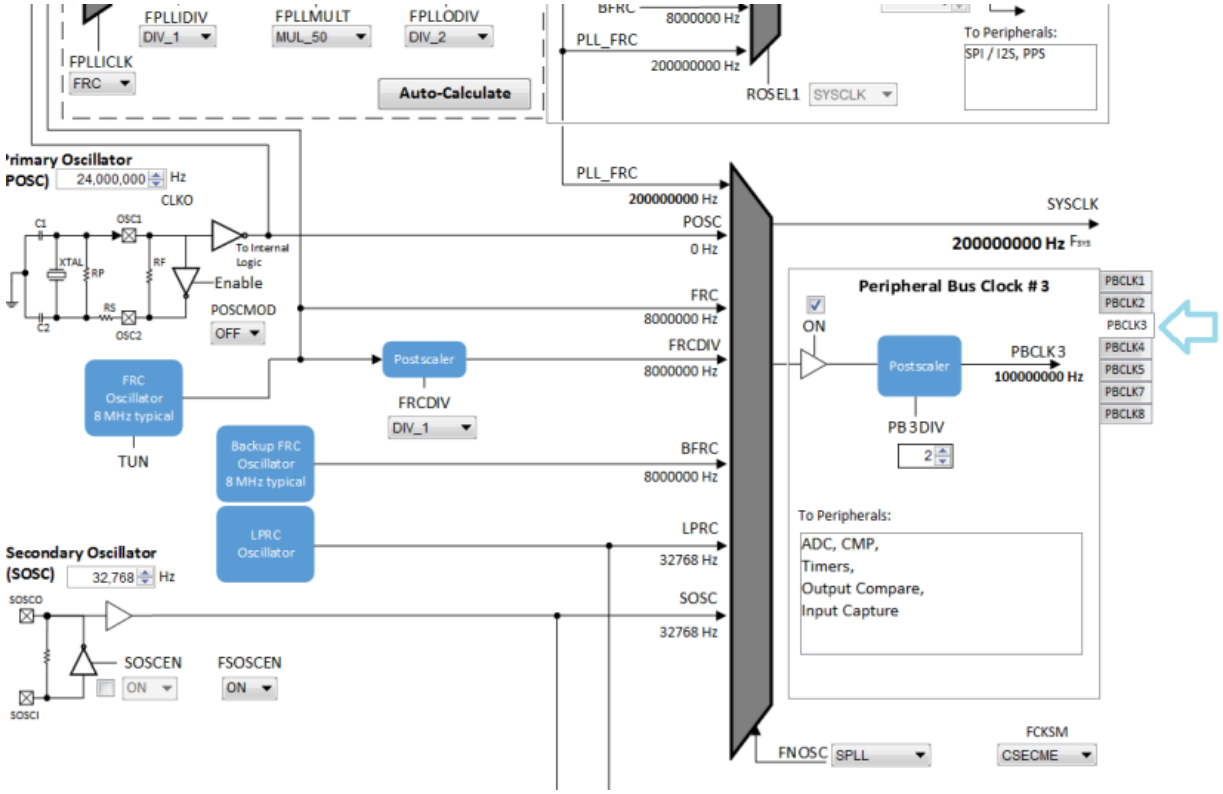


Configuring the Peripheral Bus Clocks

Provides information on configuring the peripheral bus clocks for PIC32MZ family devices.

Description

Each of the eight Peripheral Bus Clocks on the PIC32MZ family devices can be configured by using the tabs on the left.



The output frequency is in **bold**. The “To Peripherals” window provides a reminder of which peripherals each clock is driving.

This value (in Hz) corresponds to Peripheral Bus Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.

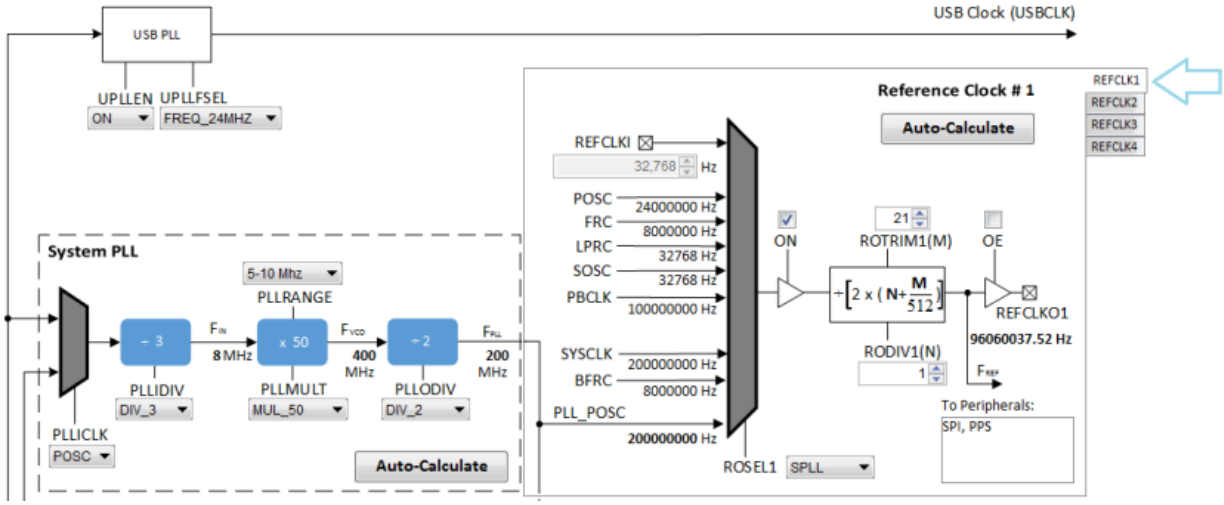
Note: It is important to know the acceptable clock range for the peripherals. The Clock Configurator will NOT provide a warning if the output peripheral clock frequency falls outside of the specified range of the peripheral.

Configuring the Reference Clocks

Provides information on configuring the reference clocks for PIC32MZ family devices.

Description

Each of the four Reference Clocks on the MZ Family of device can be configured by using the tabs on the left.



The clock input source (ROSELx), divider (RODIVx), trim value (ROTRIMx) are independently configurable. The output frequency (REFCLKOx) is in **bold**.

This value (in Hz) corresponds to Reference Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in the MHC Harmony & Application Configuration tree view.

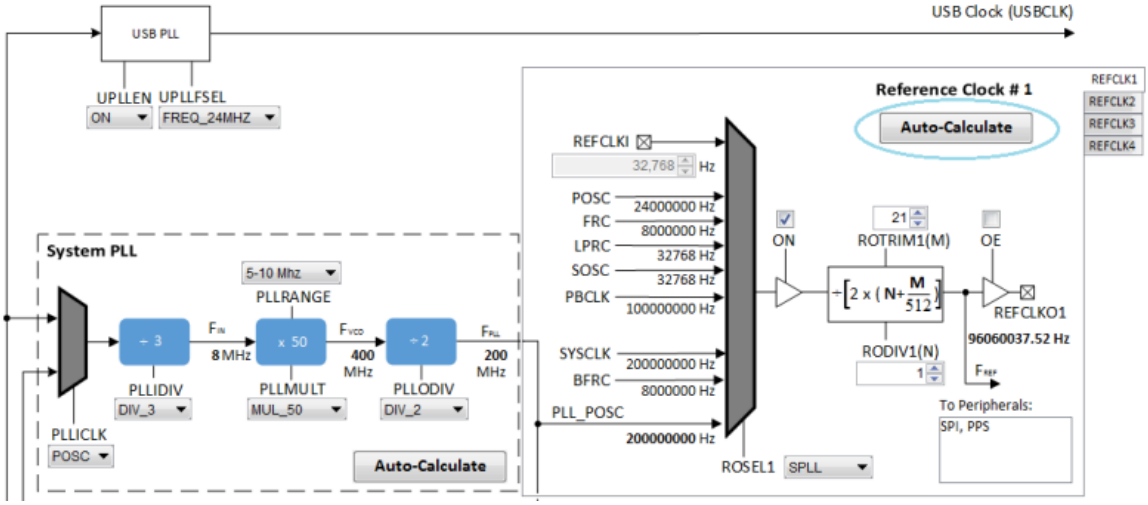
Using the Reference Clock Auto-Calculate Feature

Provides information on the reference clock auto-calculate feature for PIC32MZ family devices.

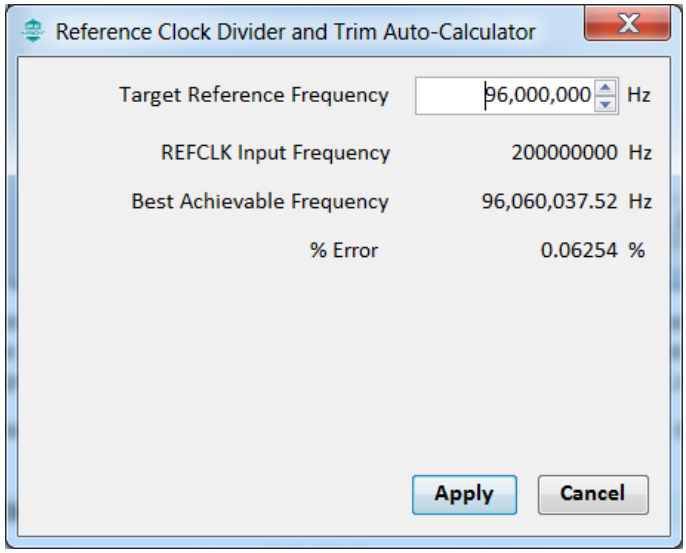
Description

The MHC Clock Configurator is equipped with the ability to help the user establish the closest possible match to a user-desired target reference clock frequency. The Auto-Calculate feature is designed to determine the divider and trim values in the each of the four reference clocks based on a user requested clock output frequency.

The feature can be accessed via the Auto-Calculate button in the Reference Clock section of the Clock Configurator.

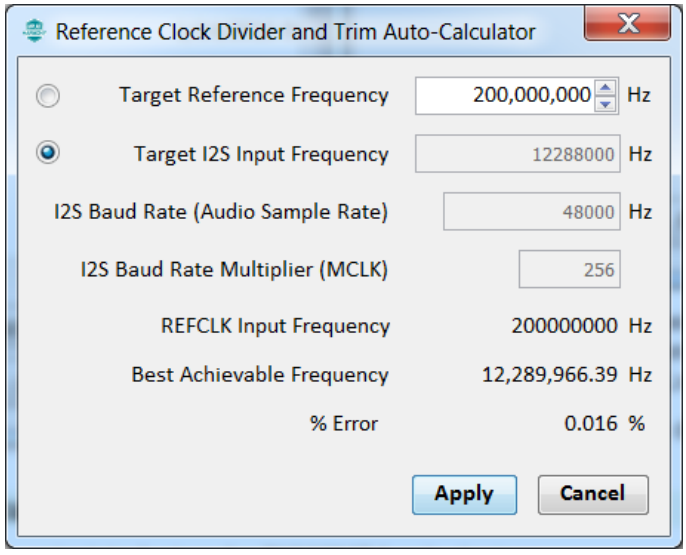


Clicking the **Auto-Calculate** button opens the Auto-Calculate dialog.



Enter the desired target reference frequency (remember to press the <Enter> key), and the dialog window will display the best achievable frequency that can be provided by the Reference Clock Divider (RODIVx) and Trim (ROTRIMx) combination, as well as the percentage discrepancy from the desired value, if any. The REFCLK Input Frequency is determined based on selection at ROSELx.

If the I2S driver is selected as part of the configuration, the Reference Clock Divider and Trim Auto-Calculator dialog opens automatically reconfigured with the option to use the target I2S input frequency as the target reference frequency.



Clicking the **Apply** button will cause the MHC Clock Configurator to update the Reference Clock divider and trim to establish the closest achievable frequency.

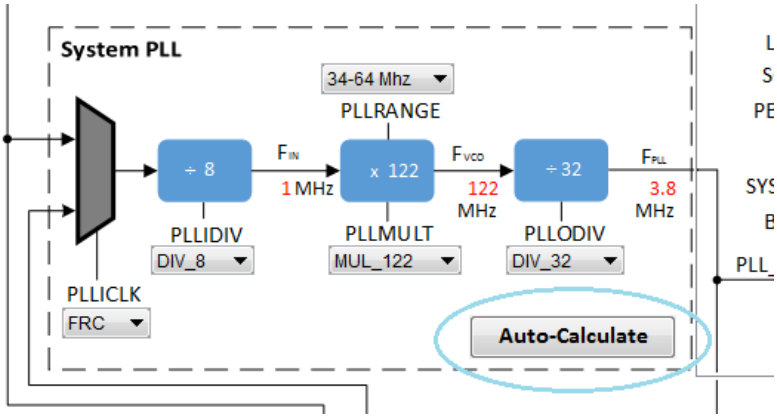
Using the SPLL Divider Auto-Calculate Feature

Provides information on the SPLL auto-calculate feature for PIC32MZ family devices.

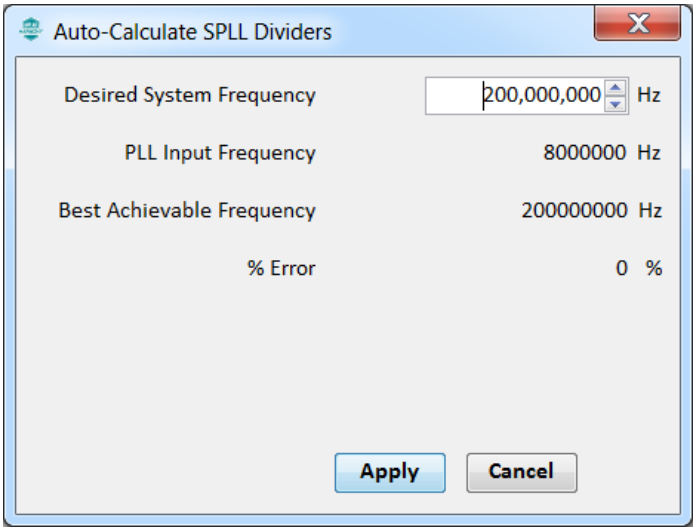
Description

The MHC Clock Configurator is equipped with the ability to help the user establish closest possible match to a user-desired target system clock frequency. The Auto-Calculate feature is designed to determine the divider and multiplier values in the SPLL-based on a user requested system clock frequency.

The feature can be accessed via the Auto-Calculate button in the SPLL section of the Clock Configurator.



Clicking the **Auto-Calculate** button opens the Auto-Calculate dialog.



Enter the desired system clock frequency (remember to press the key ENTER), and the dialog window will display the best achievable frequency that can be provided by the SPLL divider/multiplier combination, as well as the percentage discrepancy from the desired value, if any. The PLL Input Frequency is determined based on selection at PLLICLK (FRC or POSC).

Clicking the **Apply** button will cause the MHC Clock Configurator to update the SPLL dividers and multiplier to establish the closest achievable frequency.

Note: The Auto-Calculate feature will also update the PLLRANGE setting to satisfy the necessary FIN frequency.

Clock Configuration for PIC32MX Family Devices

Provides configuration information for PIC32MX family devices.

Description

The MHC Clock Configurator's support of configuring the Oscillator Module of a MX Family Device is divided into the follow sub-sections:

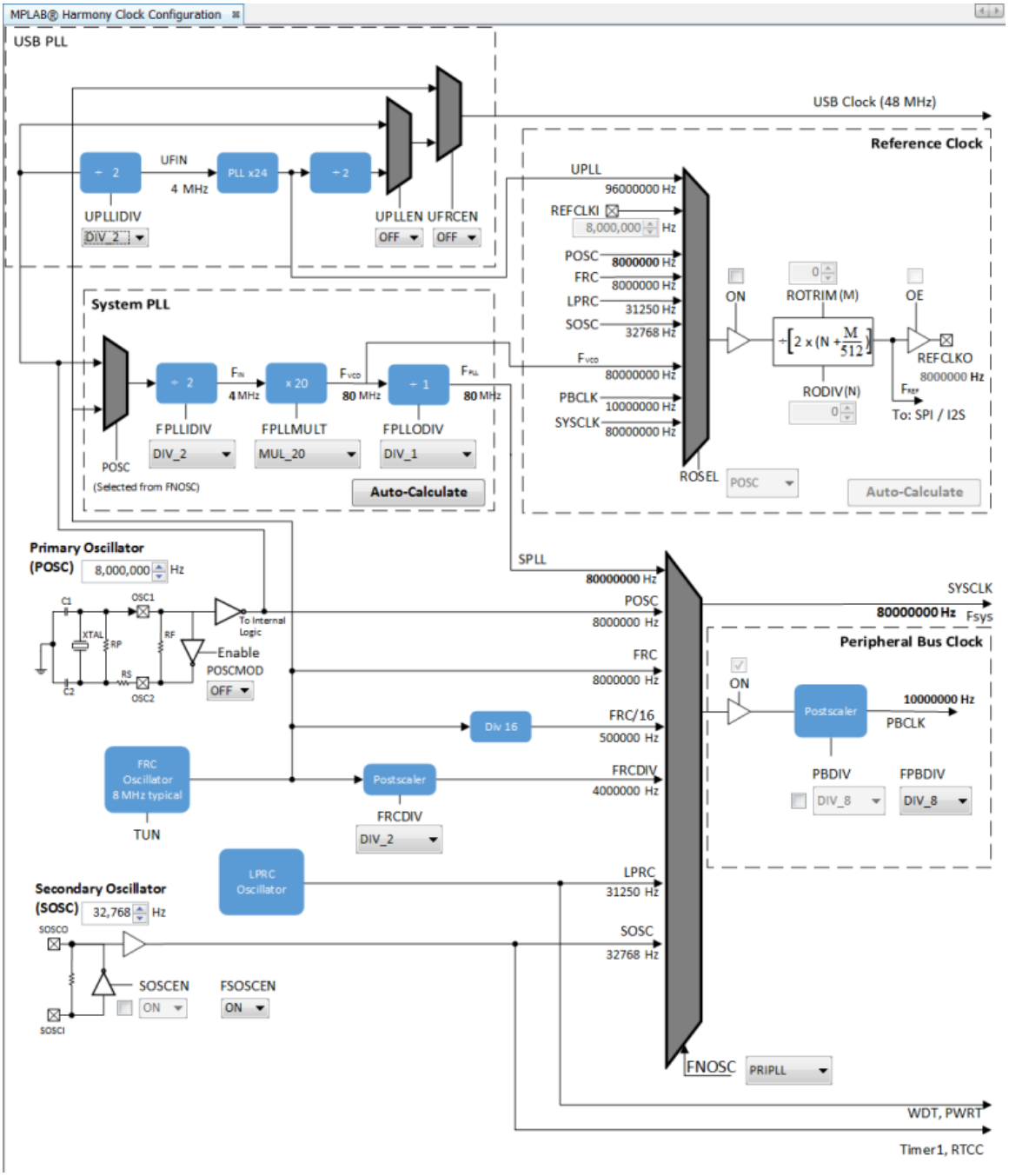
- Configuring the System Clock Frequency
- Configuring the Peripheral Bus Clock
- Configuring the Reference Clock
- Configuring the USB PLL
- Using the SPLL Divider Auto-Calculate Feature

For details regarding the operation of the Oscillator module, refer to the **"Oscillator"** chapter in the specific PIC32MX device data sheet:

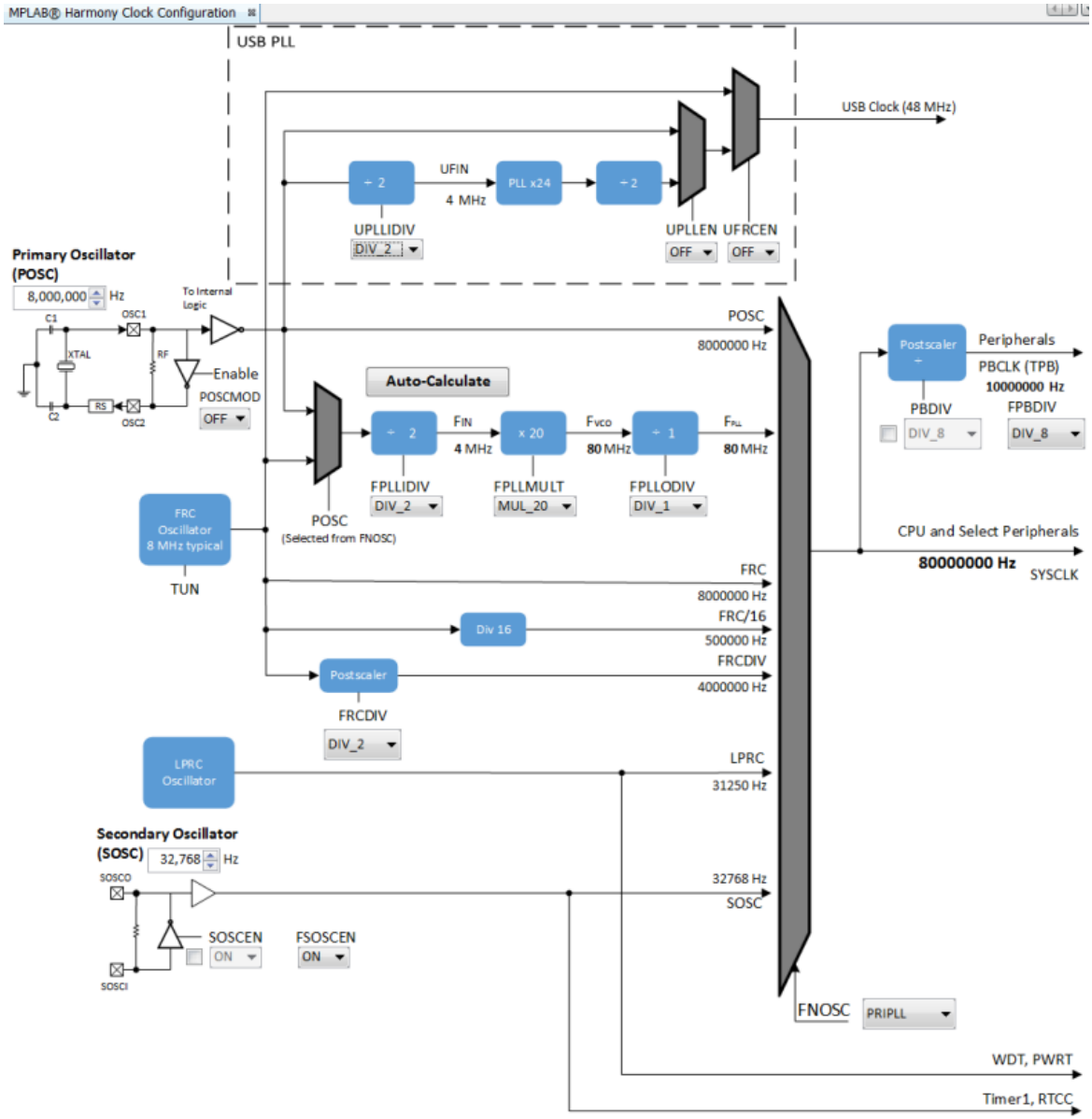
- PIC32MX1XX/2XX (DS60001168)
- PIC32MX1XX/2XX/5XX 64/100-pin Family (DS60001290)
- PIC32MX320/340/360/420/440/460 (DS60001143)
- PIC32MX330/350/370/430/450/470 (DS60001185)
- PIC32MX5XX/6XX/7XX (DS60001156)

Each of these documents are available for download from the Microchip website (www.microchip.com).

The following figure shows the configuration screen for PIC32MX1XX/2XX, PIC32MX 330/350/370/430/450/470, and PIC32MX1XX/2XX/5XX 64/100-pin Family devices.



The next figure shows the configuration screen for PIC32MX320/340/360/420/440/460 and PIC32MX5XX/6XX/7XX devices.



Configuring the System Clock Frequency

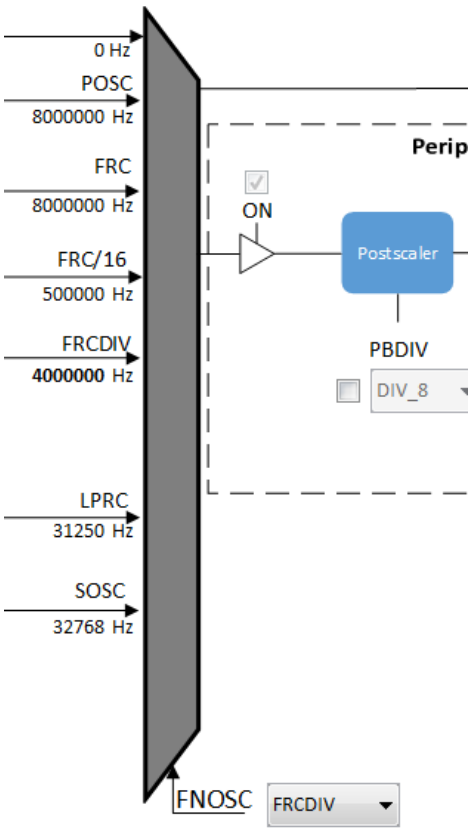
Provides information configuring the system clock frequency for PIC32MX family devices.

Description

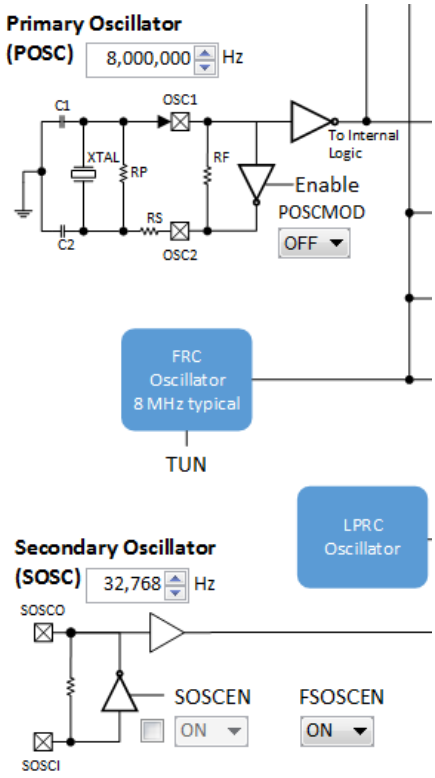
There are a total of five external and internal oscillator options as clock source:

- Internal Fast RC Oscillator (FRC) divided by the FRCDIV bits in the OSCCON register
- Internal Fast RC Oscillator (FRC) divided by 16
- Internal Low-Power RC (LPRC) Oscillator
- Secondary Oscillator (SOSC)
- Primary Oscillator with PLL module (PRIPLL)
- Primary Oscillator (POSCMOD: XT, HS, or EC)
- Internal Fast Internal RC Oscillator with PLL module via Postscaler (FRCPLL)
- Internal Fast Internal RC Oscillator (FRC)

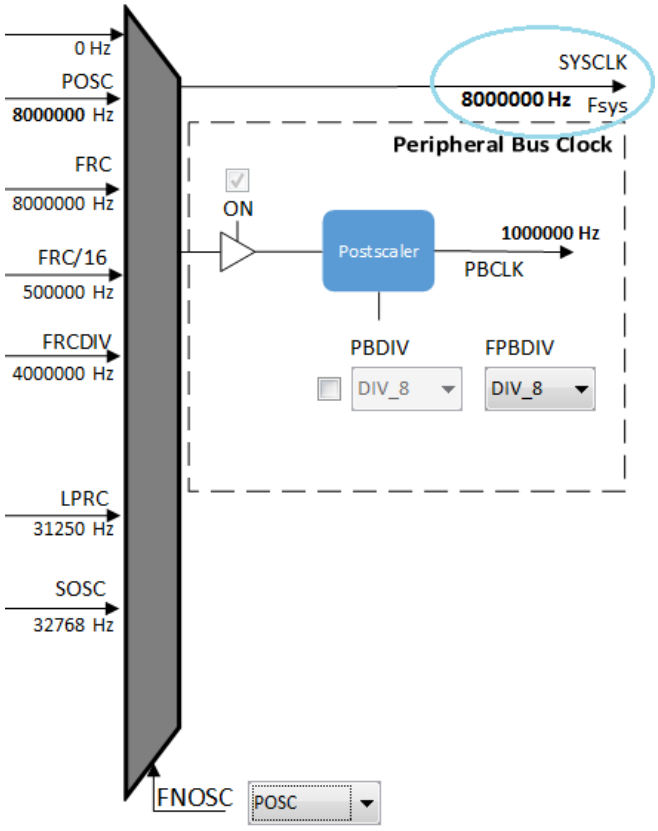
The device configuration bit FNOSC is represented as a drop-down with the above selections in the MHC Clock Configuration. The current selection is represented in **bold**.

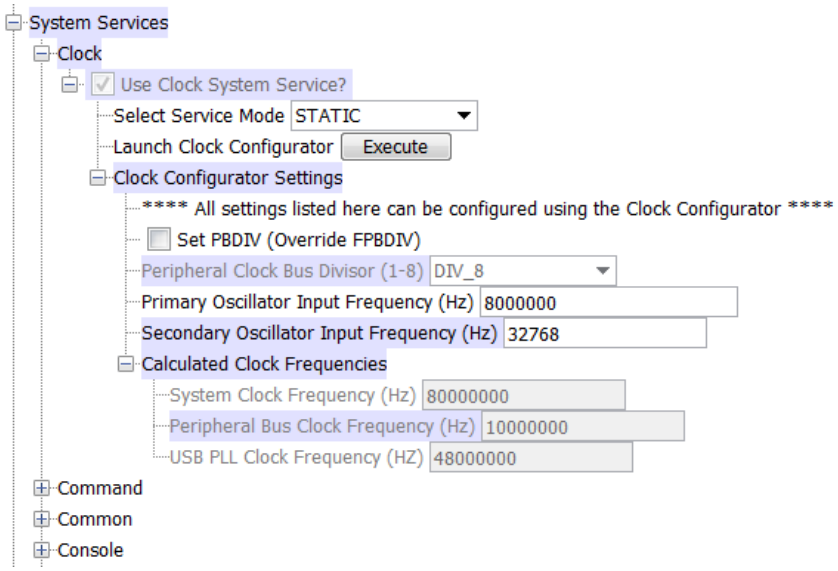


Primary Oscillator (POSC) and Secondary Oscillator (SOSC) are customizable external clock source. For POSC, the device configuration bit POSCMOD needs to be set to EC, XT, or HS. If FNOSC is set to SOSC, the device configuration bit FSOSCEN needs to be set to ON.

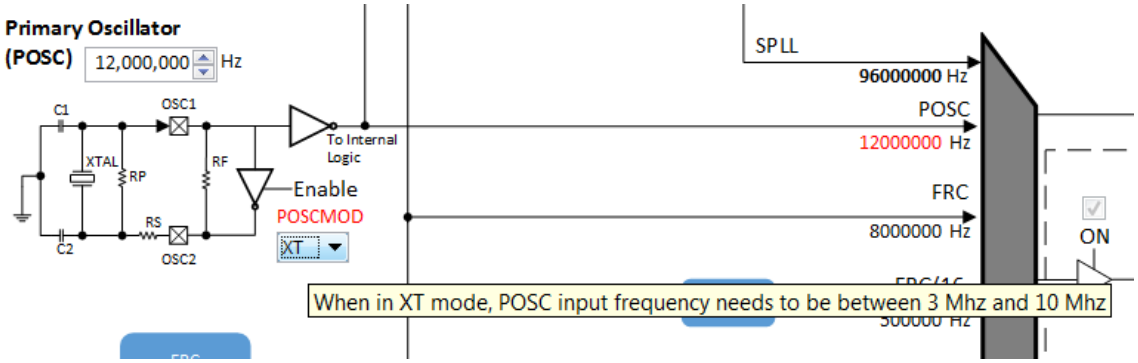


The output system clock frequency (SYSCLK) is displayed on the left side. This value (in Hz) corresponds to System Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.

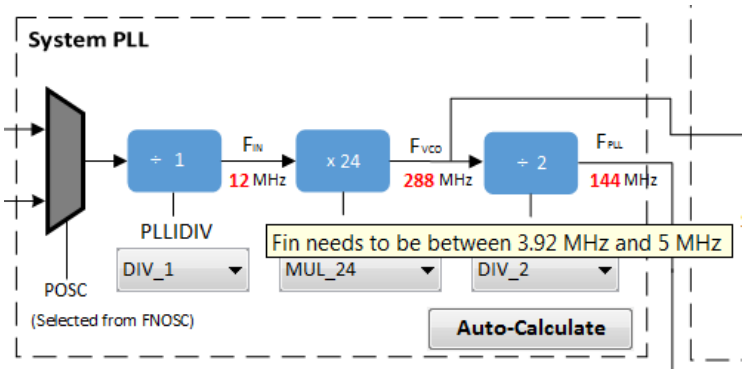




Certain frequency values may be displayed in red when the input value does not meet specification and may cripple performance of the device. An example is shown in the following figure, when the XT Oscillator Mode is selected for POSCMOD and the POSC input frequency set is outside of the 3 MHz - 10 MHz range. A dynamic help tip will also appear if the user hovers over the POSCMOD control or any of the red text.



Another example is the SPLL, where FPLL (40 MHz – 120 MHz), FVCO (60 MHz – 120 MHz), and FIN (3.92 MHz – 5 MHz) will appear in red text, including an explanation tool tip, if they fall outside of their respective required ranges.

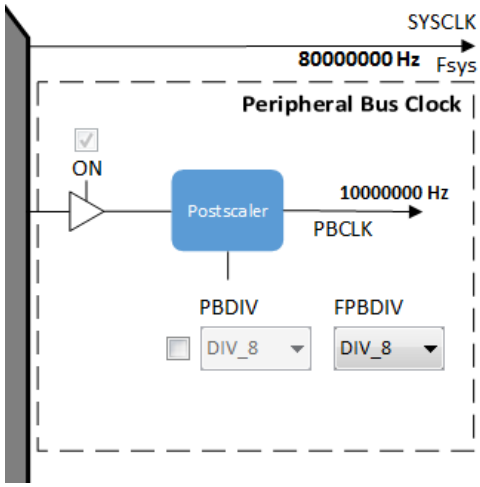


Configuring the Peripheral Bus Clock

Provides information on configuring the peripheral bus clock for PIC32MX family devices.

Description

The Peripheral Bus Clock on the MX Family of device can be configured on the left.



The output frequency is in **bold**. This value (in Hz) corresponds to Peripheral Bus Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.

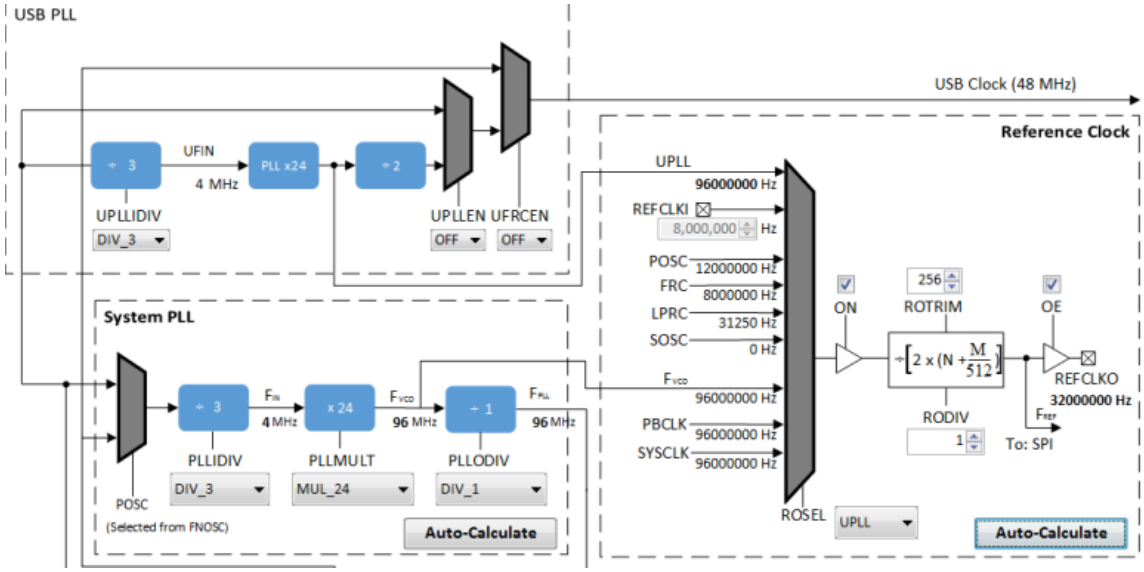
Note: It is important to know the acceptable clock range for the peripherals. The Clock Configurator will NOT provide a warning if the output peripheral clock frequency falls outside of specified range of the peripheral.

Configuring the Reference Clock

Provides information on configuring the reference clock for PIC32MX family devices.

Description

The Reference Clock on the PIC32MX1XX/2XX, PIC32MX 330/350/370/430/450/470, and PIC32MX1XX/2XX/5XX 64/100-pin Family devices can be configured in the section labeled Reference Clock on the upper right area of the screen.



The clock input source (ROSEL), divider (RODIV), trim value (ROTRIM) are independently configurable. The output frequency (REFCLKO) is in bold.

This value (in Hz) corresponds to Reference Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.

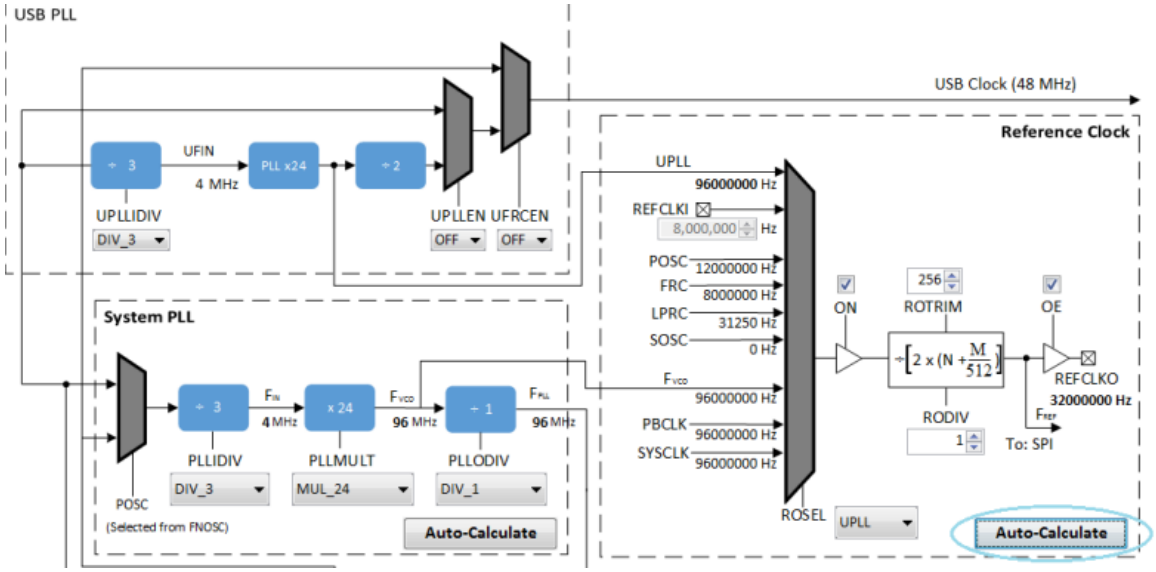
Using the Reference Clock Auto-Calculate Feature

Provides information on the reference clock auto-calculate feature for PIC32MX family devices.

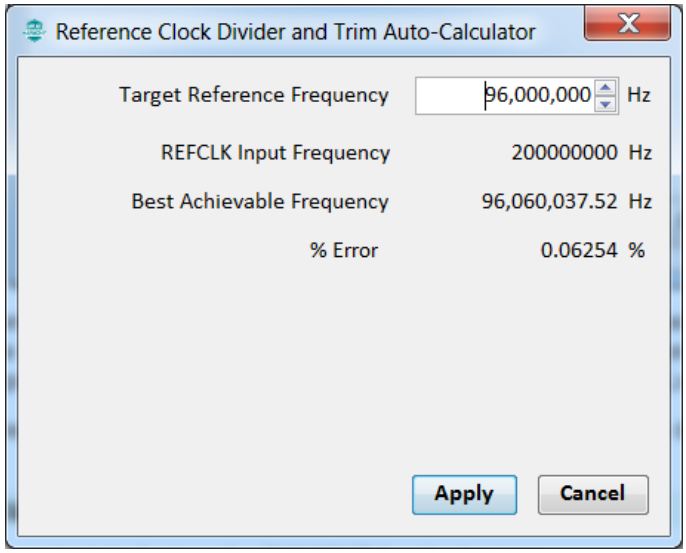
Description

The MHC Clock Configurator is equipped with the ability to help the user establish closest possible match to a user-desired target reference clock frequency. The Auto-Calculation feature is designed to determine the divider and trim values for the reference clock based on a user requested clock output frequency.

The feature can be accessed via the Auto-Calculate button in the Reference Clock section of the Clock Configurator.

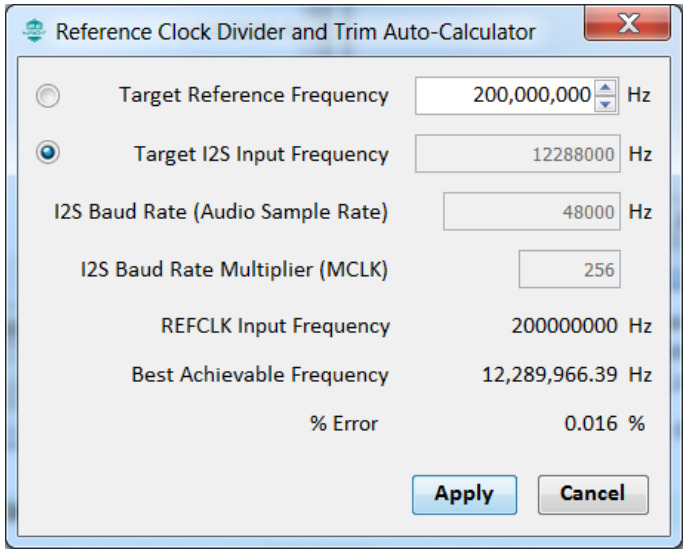


Clicking the **Auto-Calculate** button opens the Auto-Calculate dialog.



Enter the desired system clock frequency (remember to press the <Enter> key), and the dialog window will display the best achievable frequency that can be provided by the Reference Clock Divider (RODIV) and Trim (ROTRIM) combination, as well as the percentage discrepancy from the desired value, if any. The REFCLK Input Frequency is determined based on selection at ROSEL.

If the I2S driver is selected as part of the configuration, the Reference Clock Divider and Trim Auto-Calculator dialog opens automatically reconfigured with the option to use the target I2S input frequency as the target reference frequency.



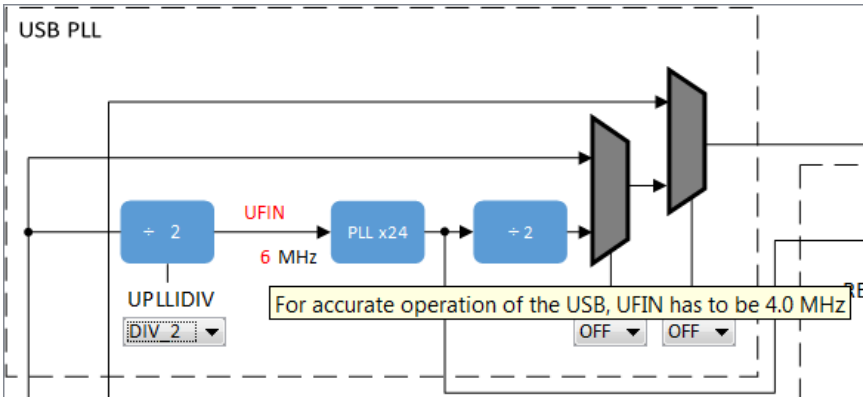
Clicking the **Apply** button will cause the MHC Clock Configurator to update the Reference Clock divider and trim to establish the closest achievable frequency.

Configuring the USB PLL

Provides information on configuring the USB PLL for PIC32MX family devices.

Description

Part of enabling the USB peripheral is to enable the USB PLL. The USB PLL requires 4 MHz input clock frequency for accurate operation. With POSC being a variable value, it is important to configure the correct USB PLL Input Divider (UPLLIDIV) value. The MHC Clock Configurator will provide visual warning if the value can lead to inaccuracy in USB operation.



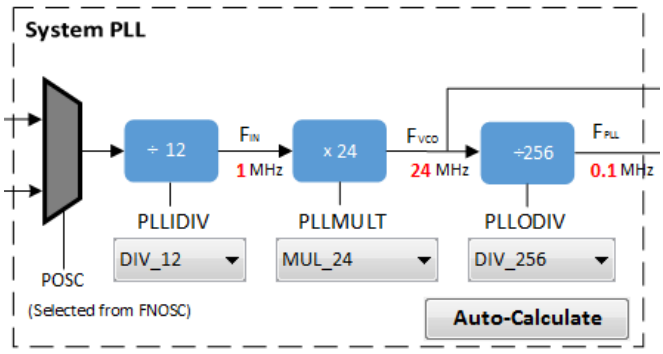
Using the SPLL Divider Auto-Calculate Feature

Provides information on using the SPLL Divider Auto-Calculate feature for PIC32MX family devices.

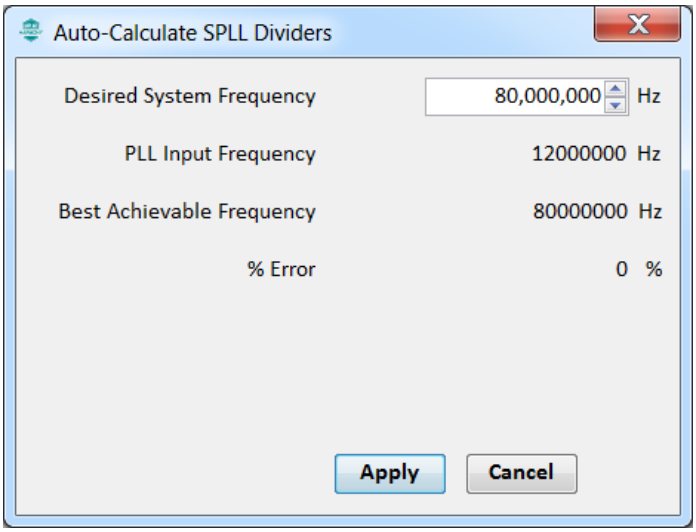
Description

The MHC Clock Configurator is equipped with the ability to help the user establish closest possible match to a user-desired target system clock frequency. The Auto-Calculation feature is designed to determine the divider and multiplier values in the SPLL-based on a user requested system clock frequency.

The feature can be accessed via the Auto-Calculate button in the System PLL section of the Clock Configurator.



Clicking the **Auto-Calculate** button opens the Auto-Calculate dialog.



Enter the desired system clock frequency (remember to press the <Enter> key), and the dialog window will display the best achievable frequency that can be provided by the SPLL divider/multiplier combination, as well as the percentage discrepancy from the desired value, if any. The PLL Input Frequency is determined based on selection at FNOSC (FRCPLL or PRIPLL).

Clicking the **Apply** button will cause the MHC Clock Configurator to update the SPLL dividers and multiplier to establish the closest achievable frequency.

MPLAB Harmony Graphical Pin Manager

Provides information on the MPLAB Harmony Graphical Pin Manager tool that resides within MHC.

Description

This graphical management tool exists for the purpose of enabling users to configure the pins of Microchip devices in a fast and intelligent manner. The tool consists of a graphical representation of the state of the component and table that provides the means to configure the pins of the device. Users intending to use this tool should be familiar with the MPLAB Harmony configuration tree.

The user configures a device using the following process:

- Launch the tool (if not already running)
- Add modules by enabling desired functionality in the configuration tree (e.g., USART or SPI)
- Using the pin table to “Lock” cells representing function and pin pairings
- Using the pin flag management dialog to change pin register values
- Generating resultant code through the **Generate** button

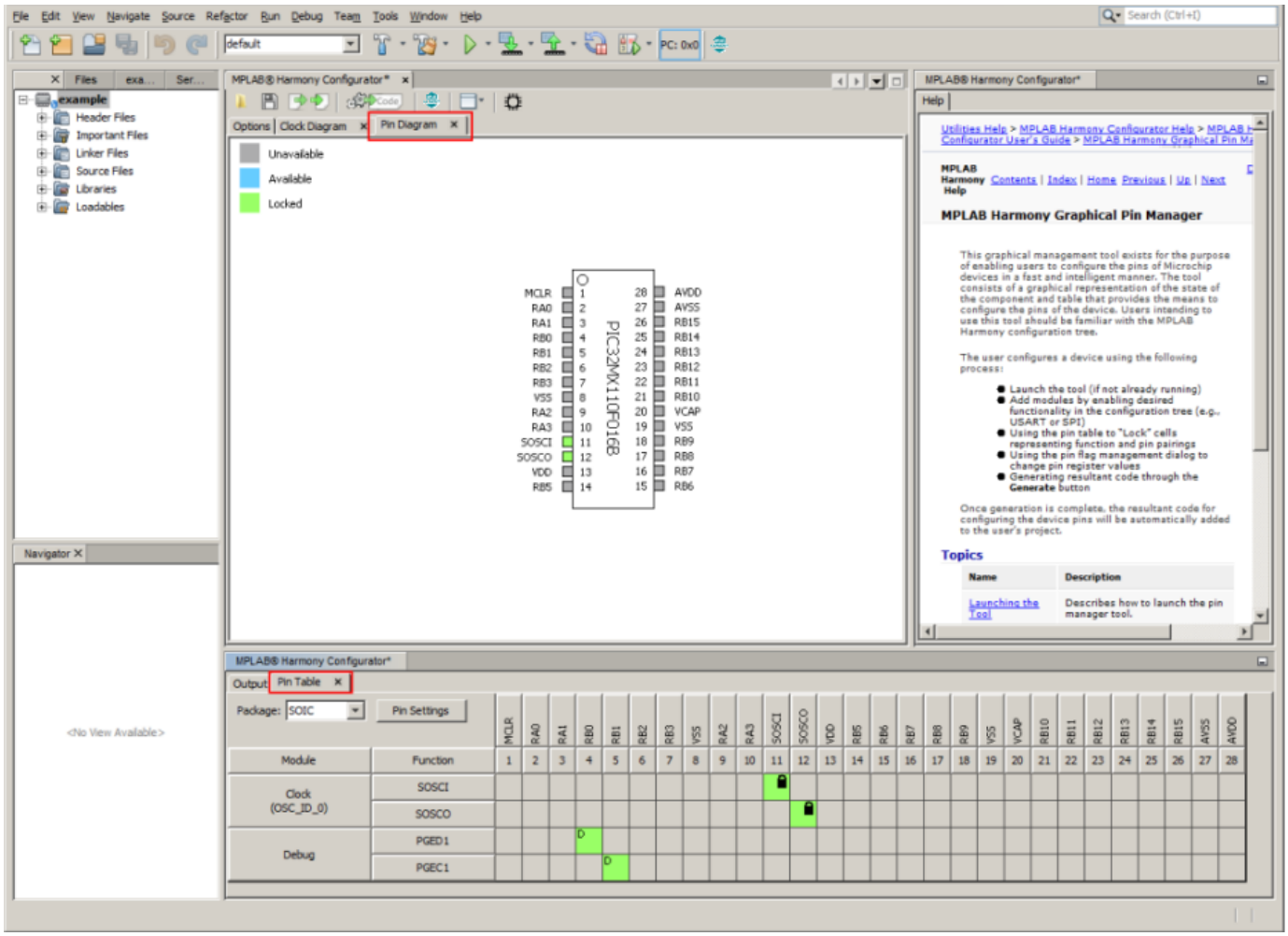
Once generation is complete, the resultant code for configuring the device pins will be automatically added to the user’s project.

Launching the Tool

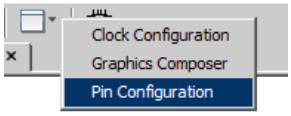
Describes how to launch the pin manager tool.

Description

The pin manager tool automatically launches when MHC starts.



The pin manager tool can be launched from the main window toolbar application launcher or from the option tree.

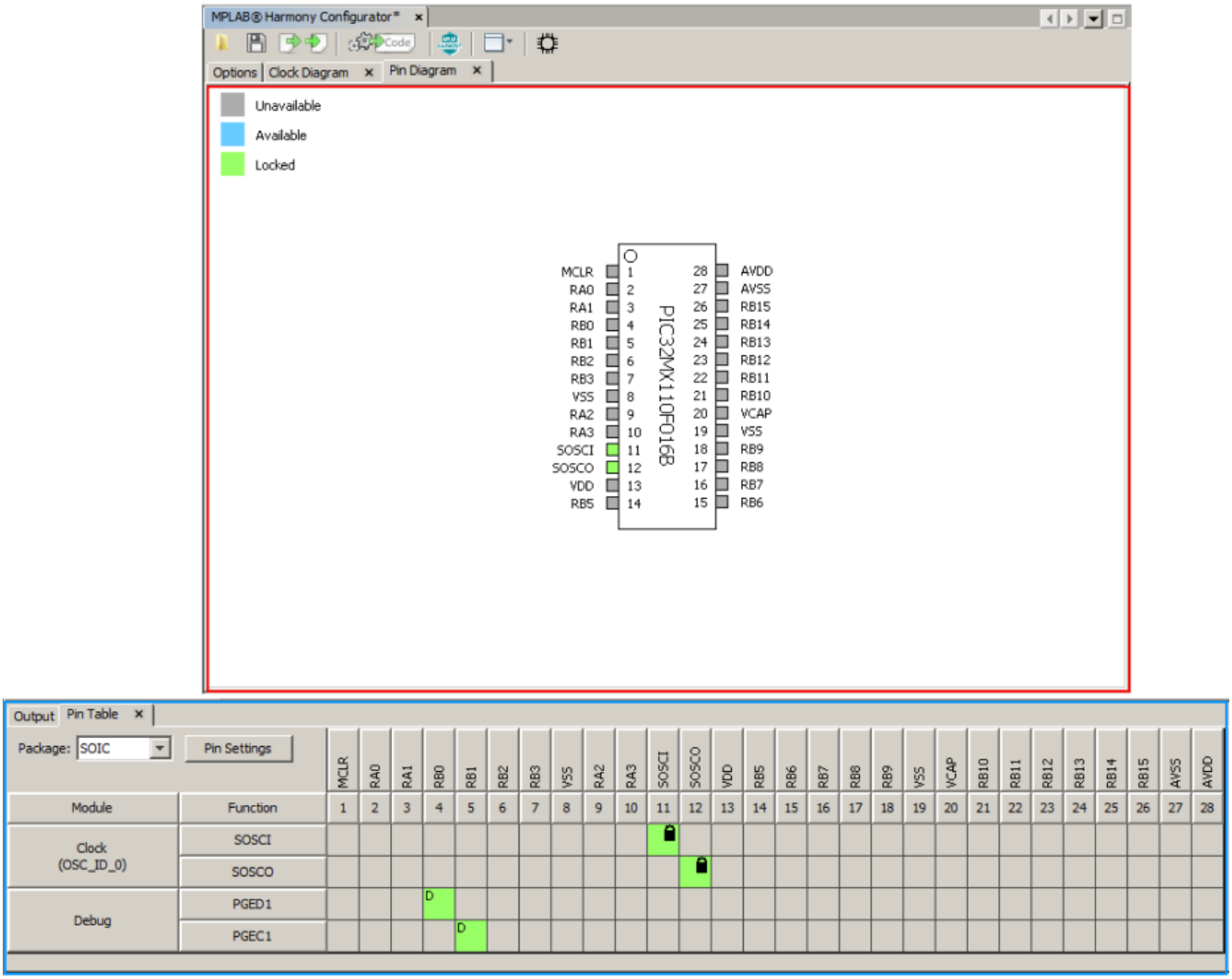


The pin manager tool can also be launched from the configuration tree.

Tool Tabs

The pin manager tool has two tabs:

- Pin Diagram (see the red section in the following figure)
- Pin Table (see the blue section in the following figure)



Pin Diagram Tab

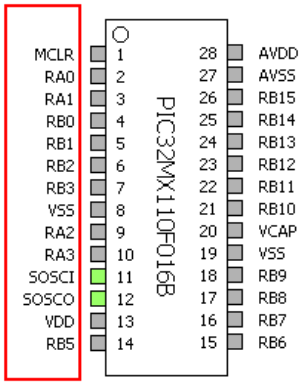
Describes the pin diagram features.

Description

This diagram is a graphical representation of the selected component to be configured. The diagram contains the following:

Pin Names

These are the base names of each pin. These names will change based on the selected function for this pin.

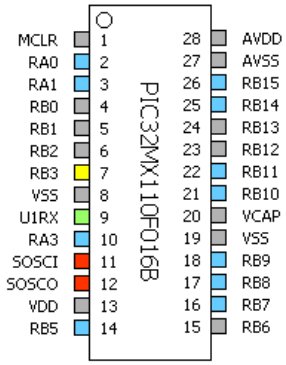


Pin States

This is a graphical indication of the state of the pin.

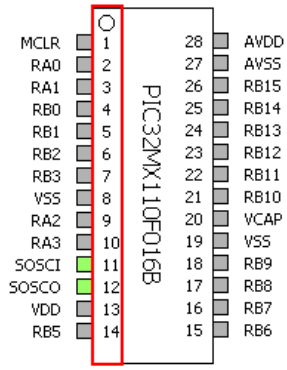
Pin States Legend:

Color	Icon	Description
Blue		This pin can be locked to an available function in the table.
Gray		This pin is currently unavailable based on the state of the pin table.
Green		This pin has been locked to a function.
Red		This pin has been automatically locked to a pin based on function priority.
Yellow		This pin is currently highlighted by the cursor.



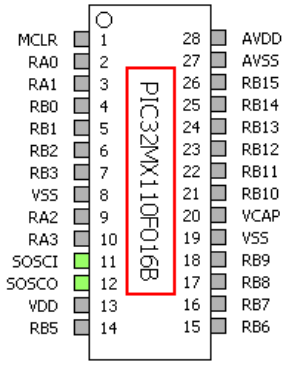
Pin Numbers

The number for each pin.



Component Name

The name of this component.



Pin Table Tab

Describes the pin table features.

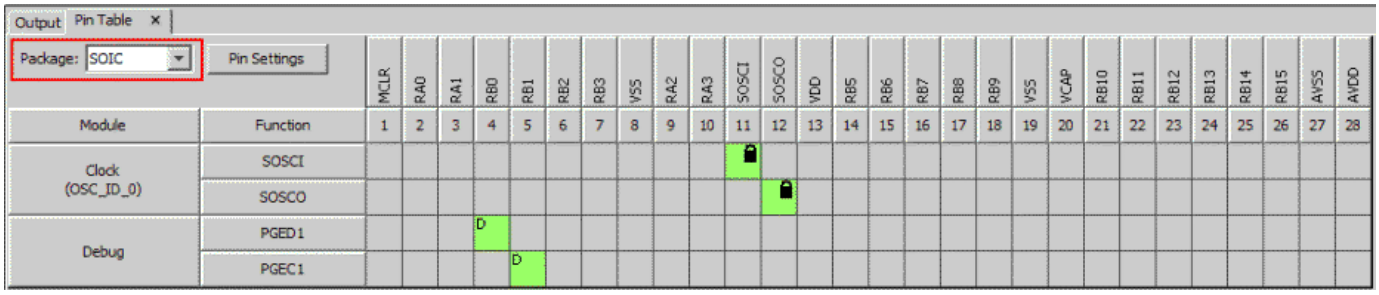
Description

The pin table allows the user to graphically configure the pins for the given component. The table contains the following areas of interest:

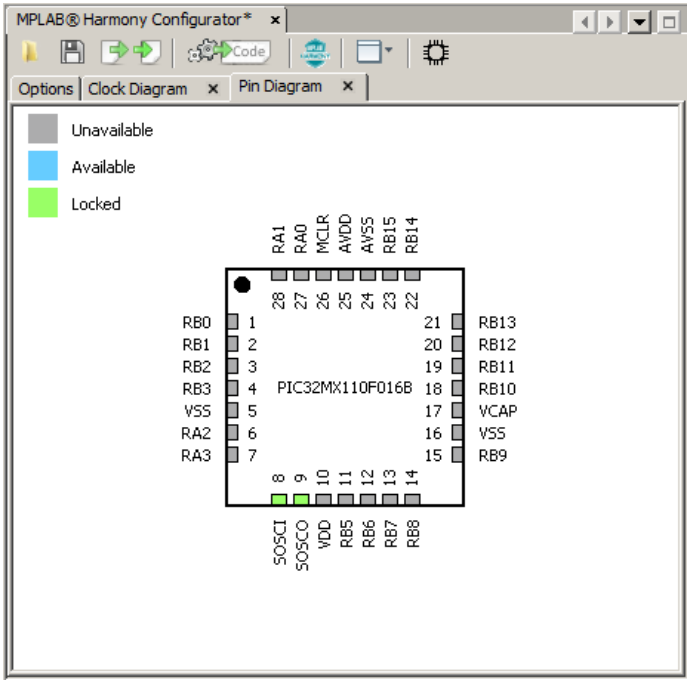
Package Selector

This menu contains the available packages for the selected component.

Note: Changing this value will reset the state of the pins to default.



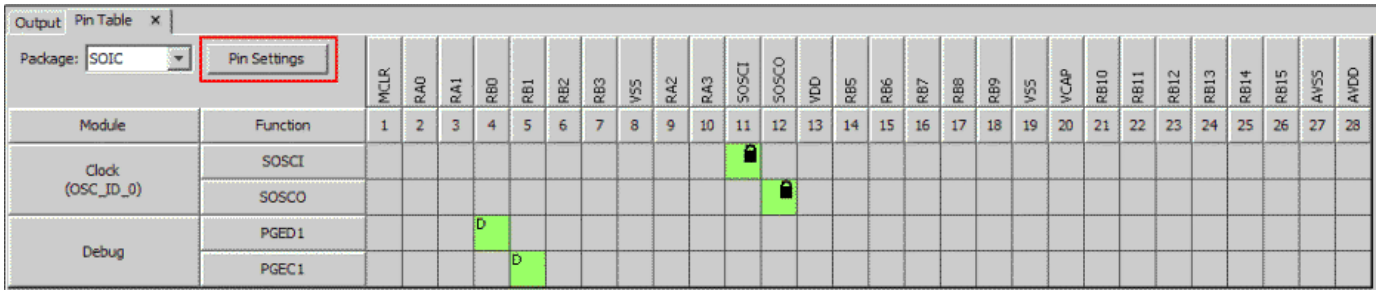
Observe the changes in the diagram and table when the QFN package is selected for this device.



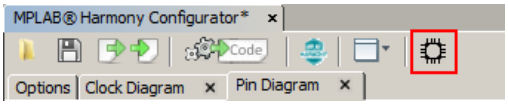
Pin Settings Button

This button shows the pin settings configuration menu. This dialog allows for the configuration of pin direction, drain, mode, latch, change notification, and pull-up and pull-down options.

Note: The direction and mode options are dependent on the function that is assigned to the pin. Board Support Package functions may lock other options as well.



The pin settings dialog can also be launched from the main toolbar when the pin diagram is visible.

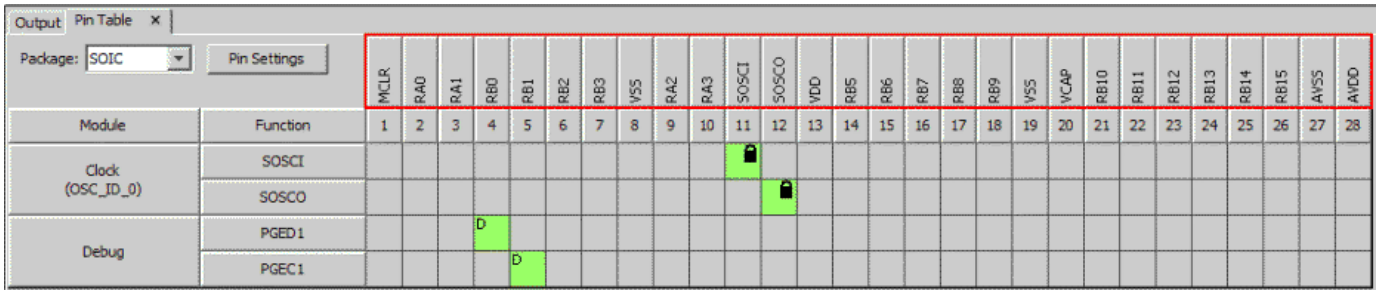


Pin Setting Configuration

Pin	Name	Voltage Tolerance	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ANSEL)	Change Notification (CNEN)	Pull Up (CNPU)	Pull Down (CNPD)
1	MCLR	5V		In	n/a	☐	Digital	☐	☐	☐
2	RA0			In	n/a	☐	Analog	☐	☐	☐
3	RA1			In	n/a	☐	Analog	☐	☐	☐
4	RB0			In	n/a	☐	Analog	☐	☐	☐
5	RB1			In	n/a	☐	Analog	☐	☐	☐
6	RB2			In	n/a	☐	Analog	☐	☐	☐
7	RB3			In	n/a	☐	Analog	☐	☐	☐
8	VSS	5V		In	n/a	☐	Digital	☐	☐	☐
9	RA2			In	n/a	☐	Analog	☐	☐	☐
10	RA3			In	n/a	☐	Analog	☐	☐	☐
11	RB4		SOSCI	n/a	n/a	☐	Analog	☐	☐	☐
12	RA4		SOSCO	n/a	n/a	☐	Analog	☐	☐	☐
13	VDD	5V		In	n/a	☐	Digital	☐	☐	☐
14	RB5	5V		In	n/a	☐	Digital	☐	☐	☐
15	RB6	5V		In	n/a	☐	Digital	☐	☐	☐
16	RB7	5V		In	n/a	☐	Digital	☐	☐	☐
17	RB8	5V		In	n/a	☐	Digital	☐	☐	☐
18	RB9	5V		In	n/a	☐	Digital	☐	☐	☐
19	VSS	5V		In	n/a	☐	Digital	☐	☐	☐
20	VCAP	5V		In	n/a	☐	Digital	☐	☐	☐
21	RB10	5V		In	n/a	☐	Digital	☐	☐	☐
22	RB11	5V		In	n/a	☐	Digital	☐	☐	☐

Pin Names

This row indicates the currently selected function for each pin. If no function is selected, the default pin name is shown instead.



Pin Numbers

This row indicates the number of each pin in the table.

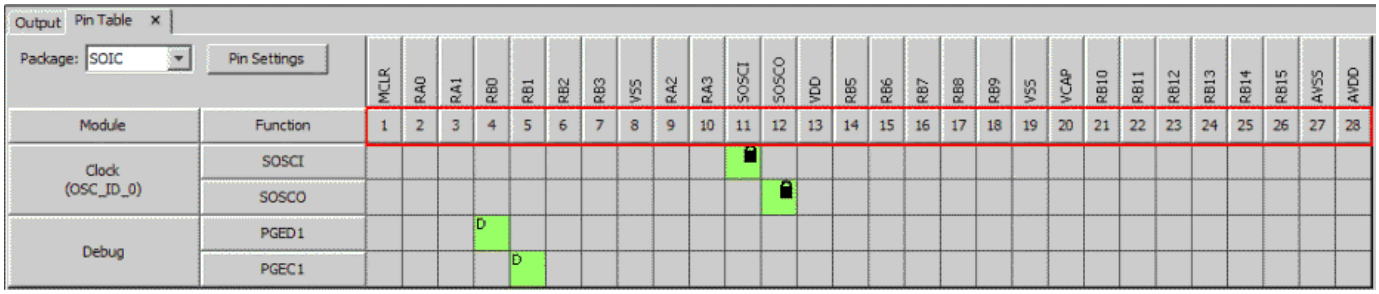


Table Modules

This column contains the modules, or groups of functions, for the current configuration. These modules are controlled by the MHC configuration tree.

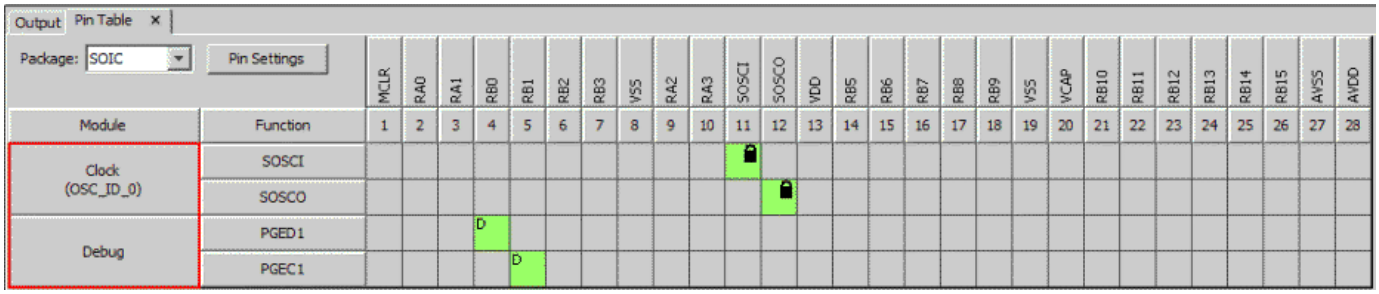


Table Functions

This column displays the functions that belong to each module.

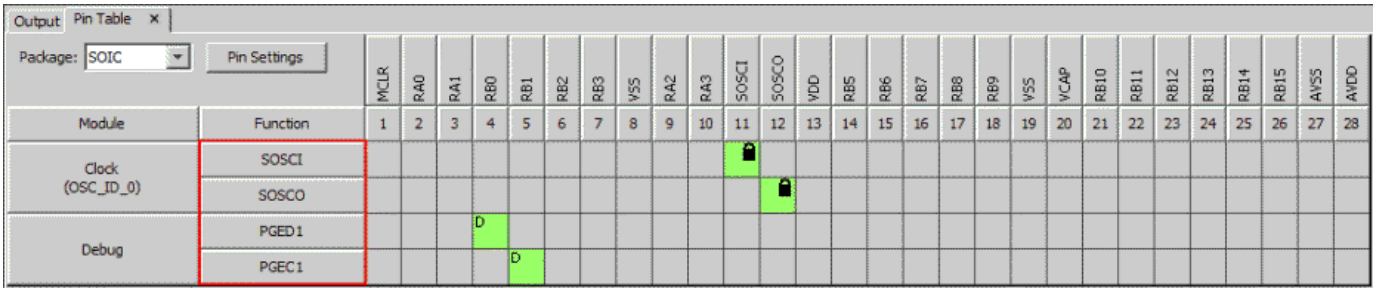


Table Grid

This area contains the grid cells. This area is for making connections between pins and functions.

Table Grid Cell Legend:

Icon	Description
	The cell is currently unavailable and cannot be selected.
	The cell is available for selection.
	The cell has been locked by the user.
D	The cell is a special debug indicator. This cell does not actually lock to a pin but is a visual debug reminder. This indicator means that the pin this cell resides on will be appropriated for debugging purposes based on the currently selected debug options.
	This cell has been automatically locked based on the available choices. This selection takes function priority into account. This lock cannot be changed by the user.

Module Management

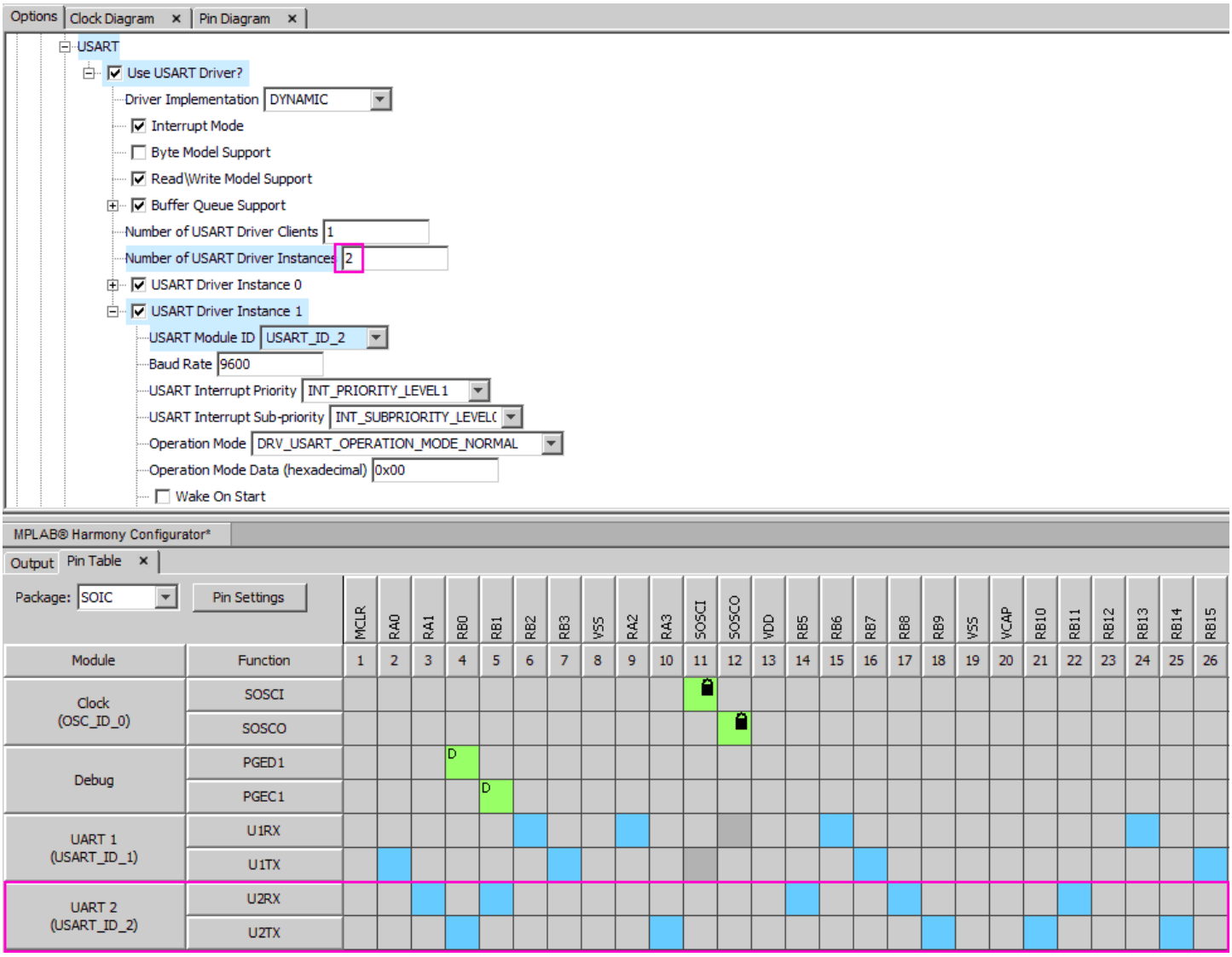
Describes the module management features.

Description

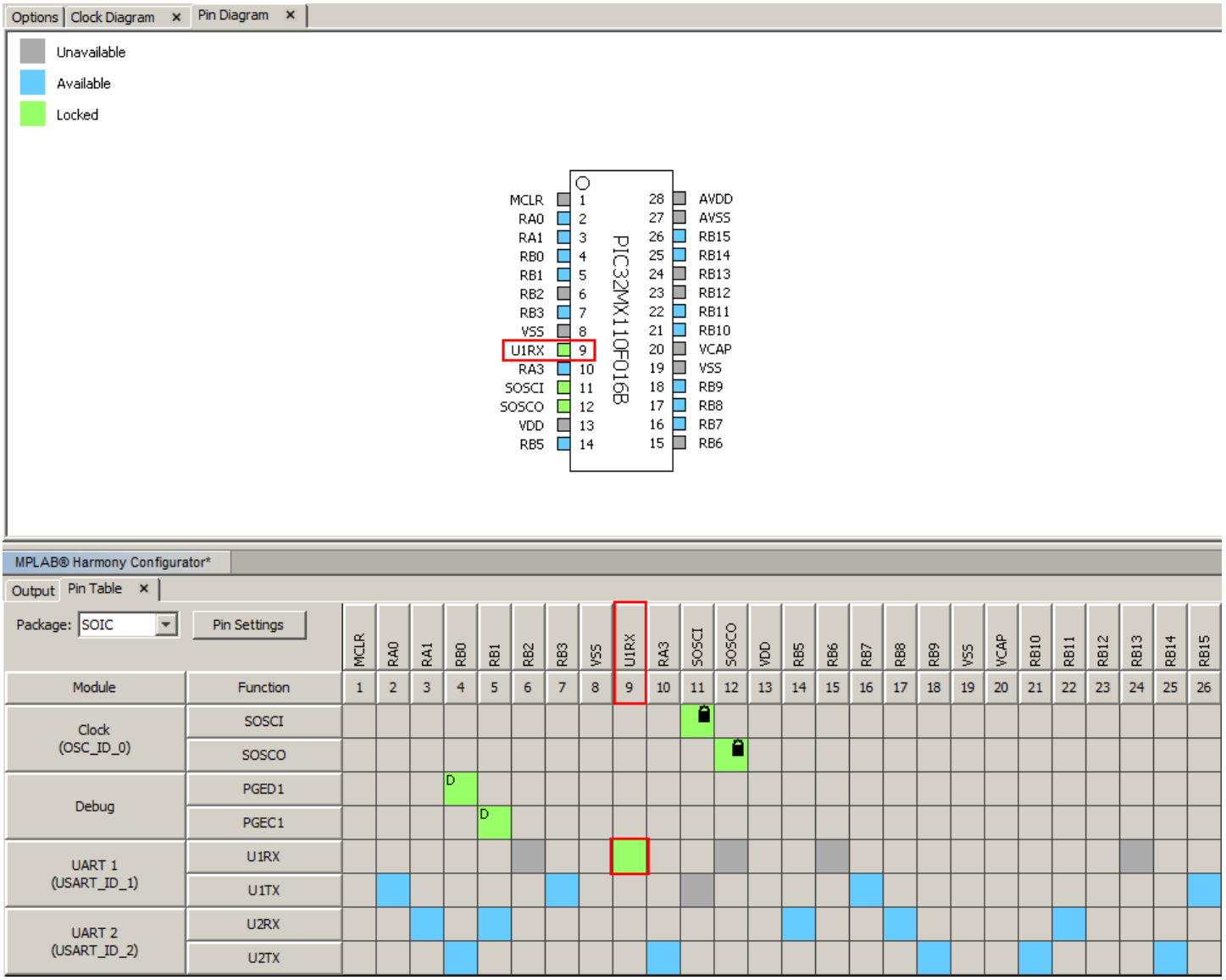
The Pin Manager table displays modules based on selections made in the configuration tree.

Observe that by enabling the USART driver instance that the USART1 module appears in the pin table.

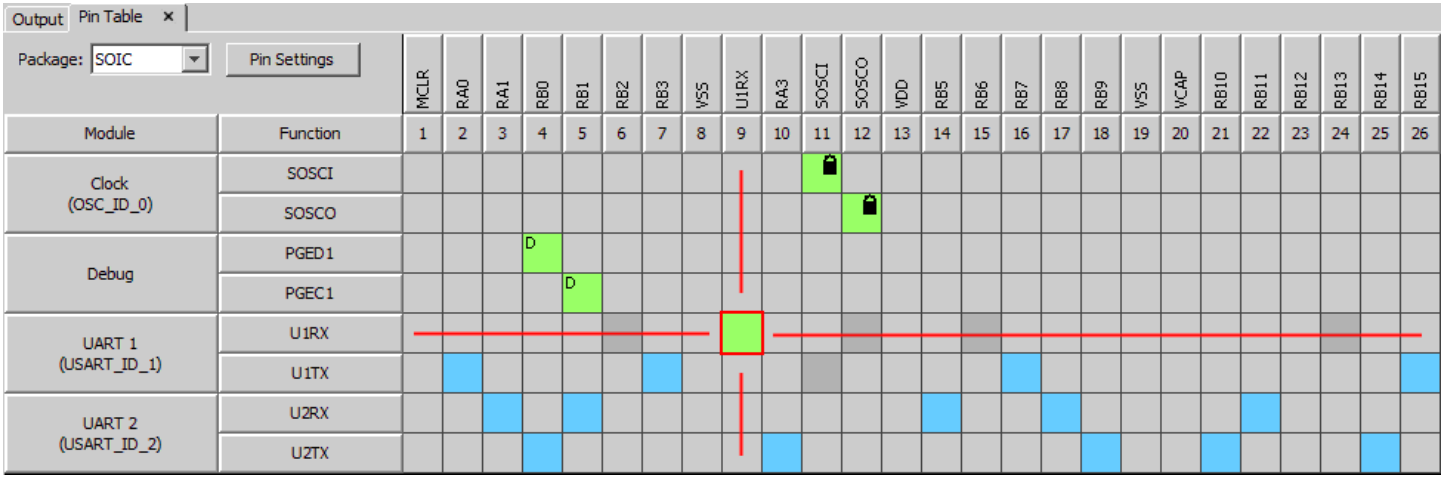
Now increase the number of USART driver instances to 2. Once the second USART instance is set to USART_ID_2, the table will display the second USART module.



The U1RX, U1TX, U2RX, and U2TX functions are Peripheral Pin Select functions and can be assigned to multiple pins. Blue cells indicate a potential pin-to-function lock. Observe that left-clicking the blue cell corresponding to pin 9 and U1RX locks that cell to that pin/function pair. U1RX is now assigned to pin 9. Observe also that the name above pin 9 has changed to indicate the locked function, as well as the name of pin 9 in the pin diagram.



With pin 9 locked, the other options for pin 9 and U1RX are now marked unavailable.



The green cell can be left-clicked again to unlock the pin and function.

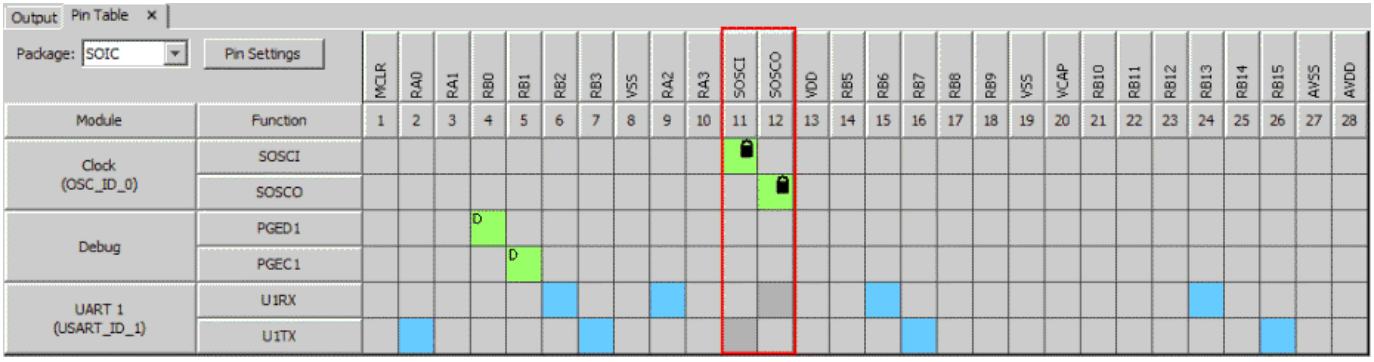
Conflict Resolution

Describes conflict resolution features.

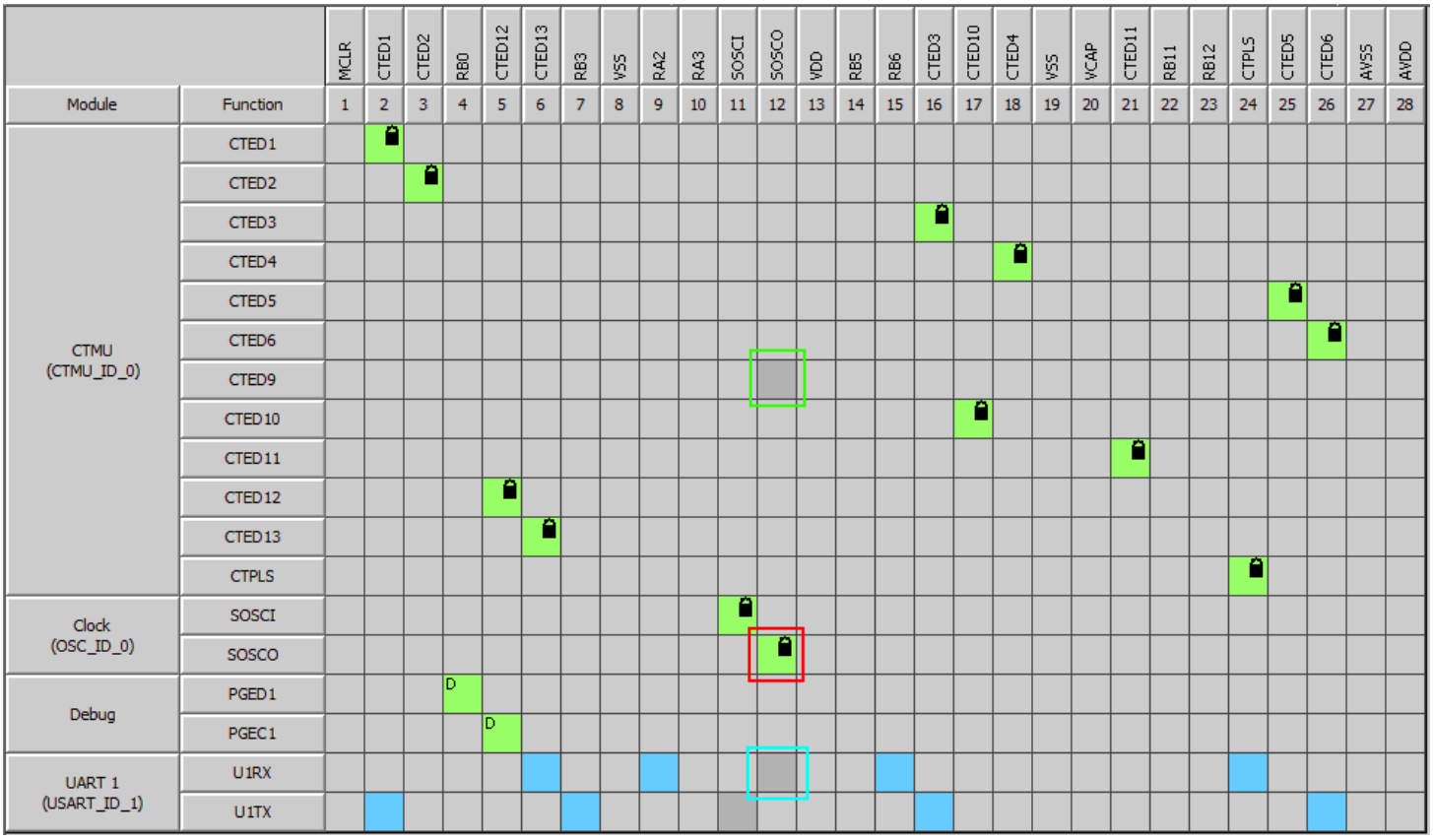
Description

The Pin Manager uses automatic conflict resolution to determine the proper function when multiple options are available.

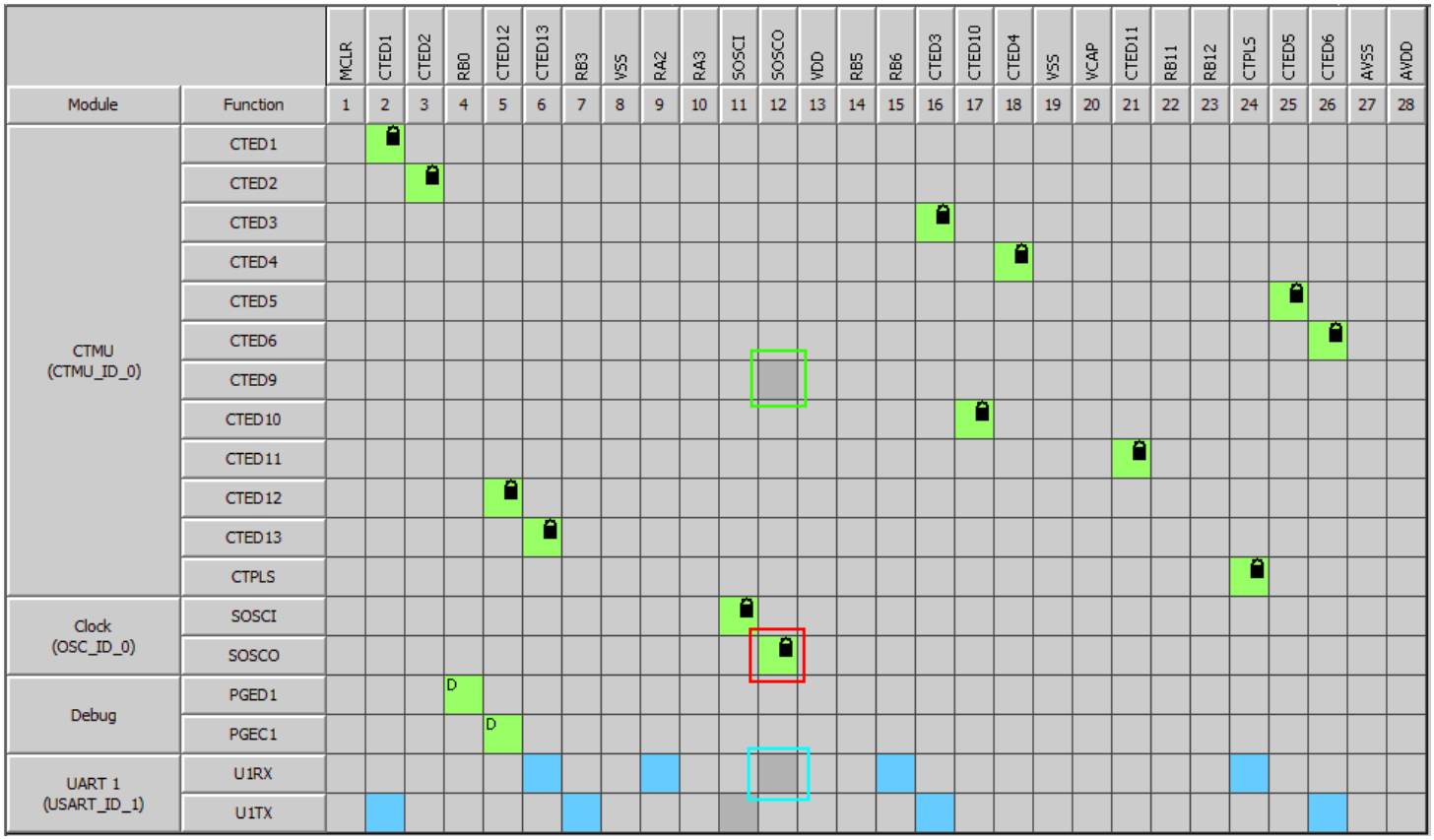
Consider the available functions for pin 12: SOSCO/RPA4/T1CK/CTED9/PMA1/RA4. Observe that the SOSCO function was given automatic priority over RPA4 (U1RX).



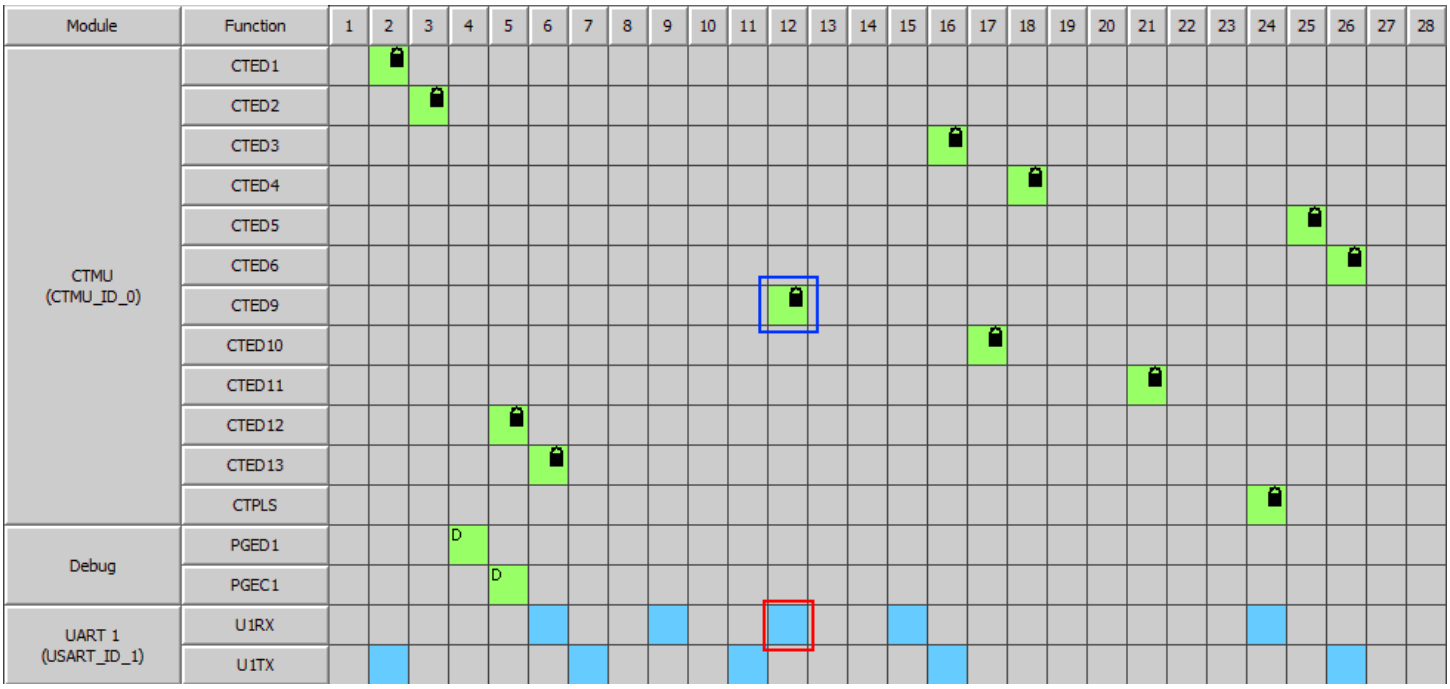
The output window displays a detailed message of this event.



Observe also that with the addition of another lower priority function that the selection does not change. The higher priority function SOSCO (red) is still automatically selected while lower priority functions RPA4 (PPS) and OC1 are disabled.



If the highest priority is a Peripheral Pin Select function (red highlight) a choice is given to the user. The next lowest priority function is automatically selected (blue highlight), but this can be overridden by user action.



If the Peripheral Pin Select function (red highlight) is manually selected then the automatic choice (blue highlight) is overridden. A conflict is still reported. If the Peripheral Pin Select function is unlocked then the lower priority function will be automatically locked again.

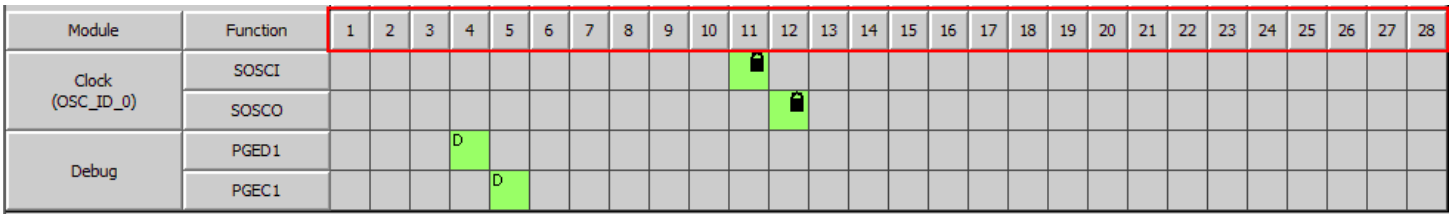
Pin Table Features

Describes pin table features.

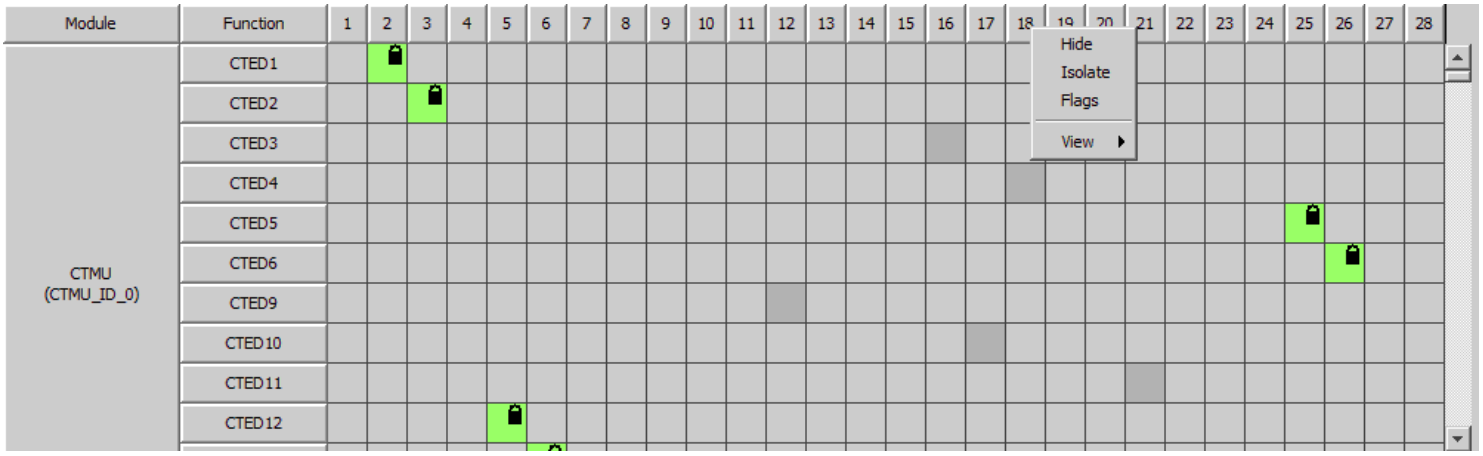
Description

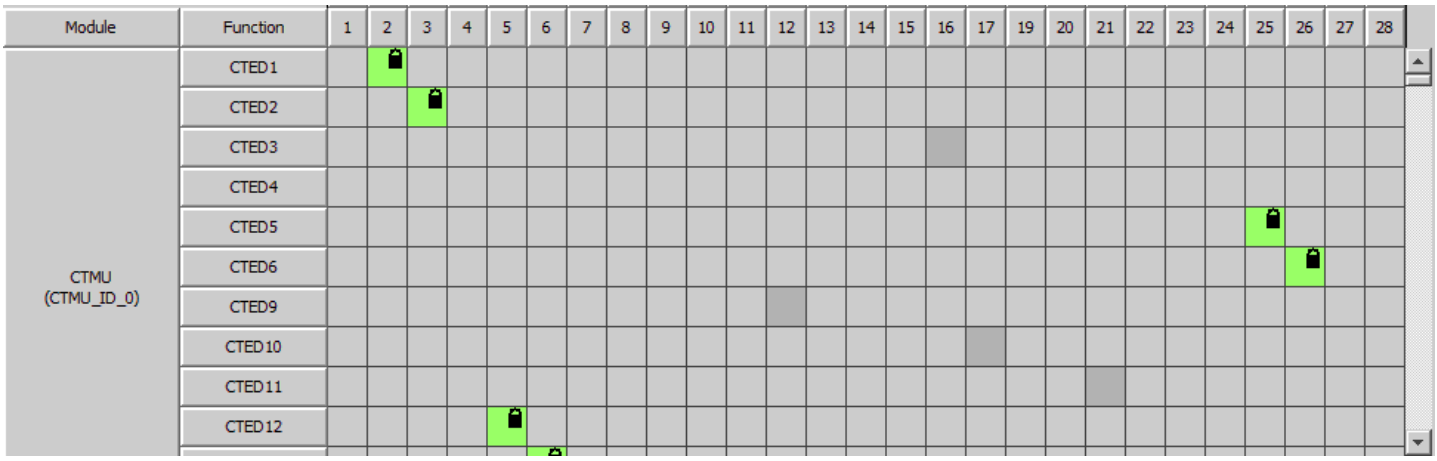
The Pin Table can be reconfigured to show as little or as much information as the user desires. For example, individual pin rows can be hidden or

isolated depending on how much information is desired. This is accomplished by right-clicking on a pin number and selecting a desired option from the context menu.

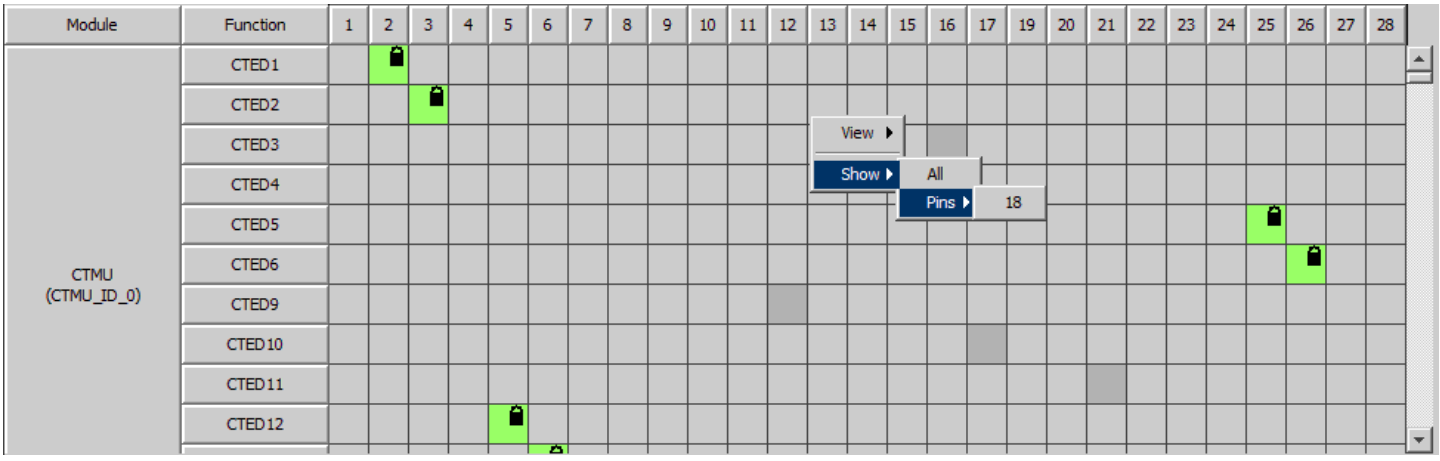


To remove pin 18 from the table, right-click the pin 18 number box. Select **Hide** from the context menu.

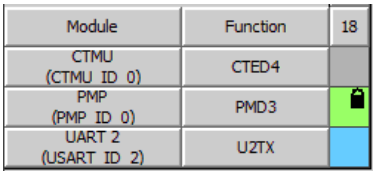




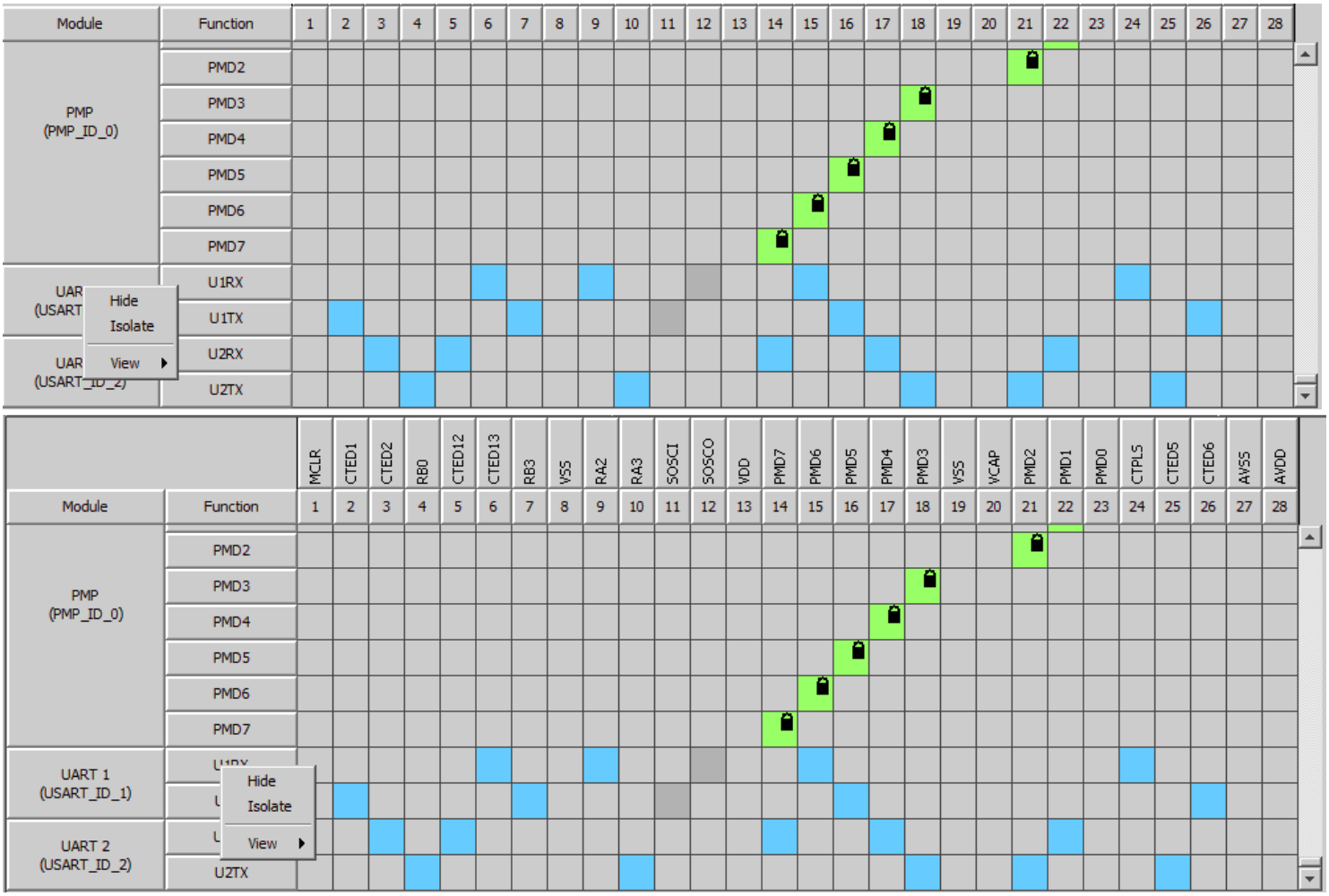
Observe that pin 18 has been removed from the table. To restore the column, right-click in the table and select **Show > All** or navigate the available sub-menus and select pin 18.



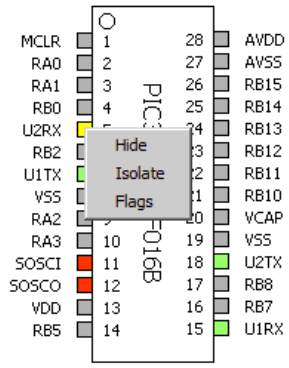
The table can also be reduced to show only desired pins and functions by using the "Isolate" command. To show only pin 18, again right-click on the pin 18 number box and select **Isolate**.



This functionality also exists for pin modules, functions, and ports.



The table can also be modified by right-clicking the pin boxes in the pin diagram.



The table can also be reconfigured to display pins according to their respective ports. To do this, right-click the table, navigate to the View sub-menu, and select **Ports**. The top row is the original pin number, the middle row shows the port grouping, and the bottom row is the pin's number inside the port grouping. Ports can also be hidden and isolated in the same manner as pins, modules, and functions. This is accomplished by right-clicking on the port name box.

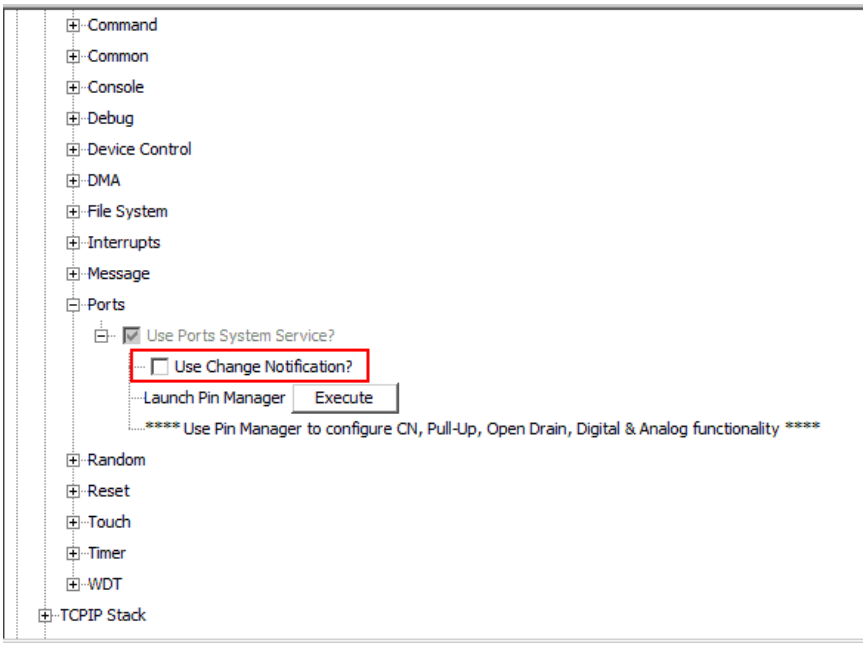
		2	3	9	10	12	4	5	6	7	11	14	15	16	17	18	21	22	23	24	25	26
		A					B															
Module	Function	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Clock (OSC_ID_0)	SOSCI																					
	SOSCO																					
Debug	PGED1						D															
	PGEC1							D														
UART 1 (USART_ID_1)	U1RX																					
	U1TX																					
UART 2 (USART_ID_2)	U2RX																					
	U2TX																					

Change Notification and Non-PPS Devices

Describes handling change notification for non-PPS devices.

Description

For non PPS parts, change notifications behave differently. They must be explicitly enabled in the configuration tree.



When enabled, the Change Notification module appears in the table. Change notification cells behave similarly to Peripheral Pin Select functions. They will be overridden by higher priority functions, but will provide a user choice if they are the highest priority.

The pin flag dialog also behaves differently for Non-PPS parts. The "Change Notification", "Pull Up", and "Pull Down" options are disabled.

Pin Setting Configuration

Pin	Name	Voltage Tolerance	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ADPCFG)	Change Notification (CNEN)	Pull Up (CNPUE)	Pull Down (CNPD)
1	RG15	5V		In	n/a		Digital			
2	VDD	5V		In	n/a		Digital			
3	RE5	5V		In	n/a		Digital			
4	RE6	5V		In	n/a		Digital			
5	RE7	5V		In	n/a		Digital			
6	RC1	5V		In	n/a		Digital			
7	RC2	5V		In	n/a		Digital			
8	RC3	5V		In	n/a		Digital			
9	RC4	5V		In	n/a		Digital			
10	RG6	5V		In	n/a		Digital			
11	RG7	5V		In	n/a		Digital			
12	RG8	5V		In	n/a		Digital			
13	MCLR	5V		In	n/a		Digital			
14	RG9	5V		In	n/a		Digital			
15	VSS	5V		In	n/a		Digital			
16	VDD	5V		In	n/a		Digital			
17	RA0	5V		In	n/a		Digital			
18	RE8	5V		In	n/a		Digital			
19	RE9	5V		In	n/a		Digital			
20	RB5			In	n/a		Analog			
21	RB4			In	n/a		Analog			
22	RB3			In	n/a		Analog			

Exporting Pin Mapping

Provides information on exporting pin mappings.

Description

The MPLAB Harmony Graphical Pin Manager provides the ability to export the pin mapping of the current configuration into Excel in `.xls` format for the purpose of printing out the pin mapping. Refer to Importing and Exporting Data for the steps to export the pin mapping.

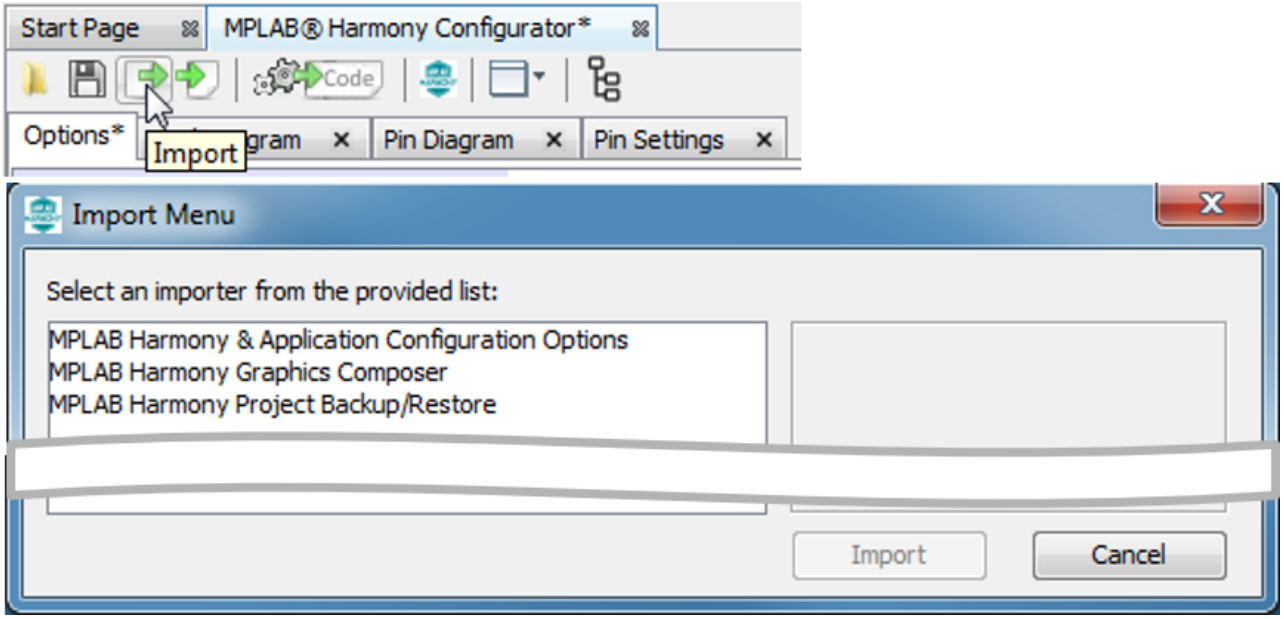
Importing and Exporting Data

Provides information on importing and exporting data to/from the MHC.

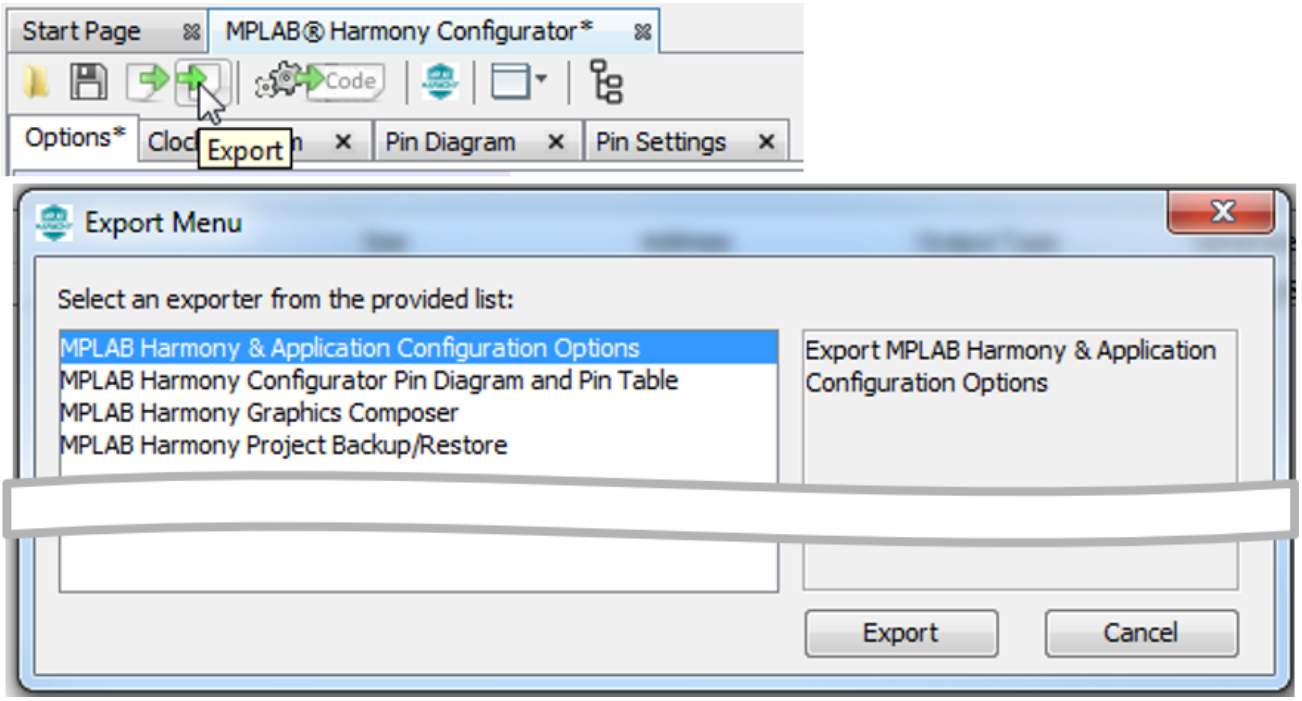
Description

The MPLAB Harmony Configurator provides several options for importing and export various types of data to and from the application. The import and export icons can be found in the main window toolbar.

The Import dialog shows the various data sources that can be imported into MPLAB Harmony Configurator. To import, select an item from the list and click **Import**. The second option, MPLAB Harmony Graphics Composer, is only available when the Graphics Composer screen is active (see the following figure).



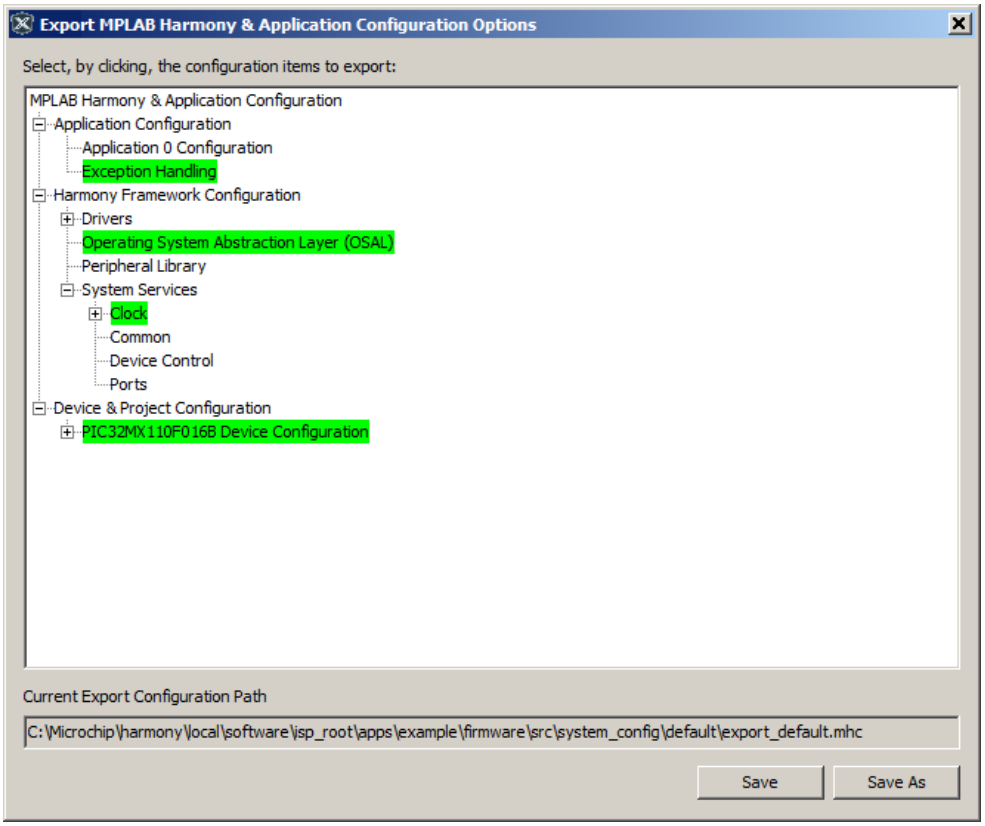
The Export dialog shows the various data sources that can be exported from MPLAB Harmony Configurator. To export, select an item from the list and click **Export**. The third option, MPLAB Harmony Graphics Composer, is only available when the Graphics Composer screen is active (see the following figure).



Importing and Exporting MPLAB Harmony Configurator Configuration Options

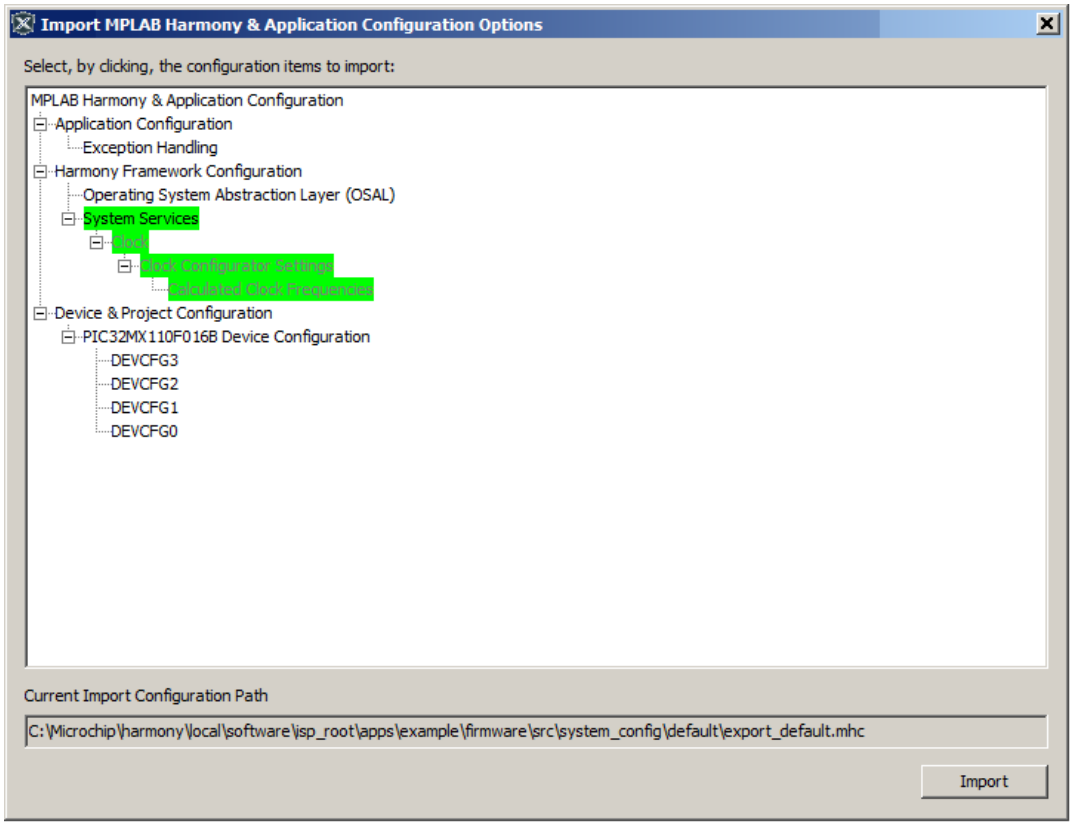
By selecting **MPLAB Harmony & Application Configuration Options** from either the Import or Export dialog, the user has the ability to create or import `.mhc` files with only user-selected options.

The following figure provides an example of the option export dialog.



To use this feature, left-click any desired option to toggle its state. Green-highlighted options will be exported. Then, use the **Save** and **Save As** buttons as desired to write the file.

To import, select the option import from the Import dialog and select the previously exported file. Observe that only the exported options are visible in the import window. The user can again select and highlight items in green to select them for import. When all desired settings have been highlighted, click **Import**.

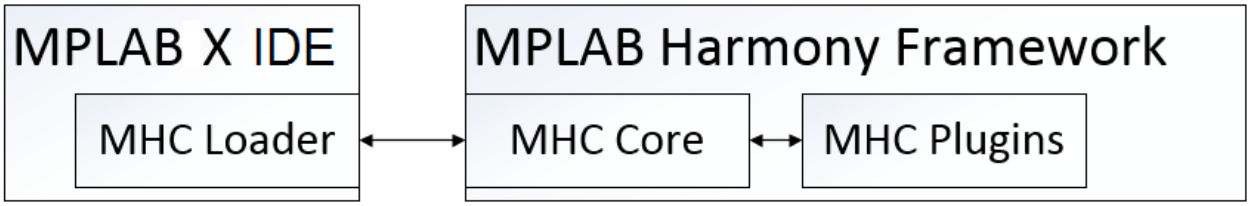


Understanding MPLAB Harmony and MHC Version Numbers

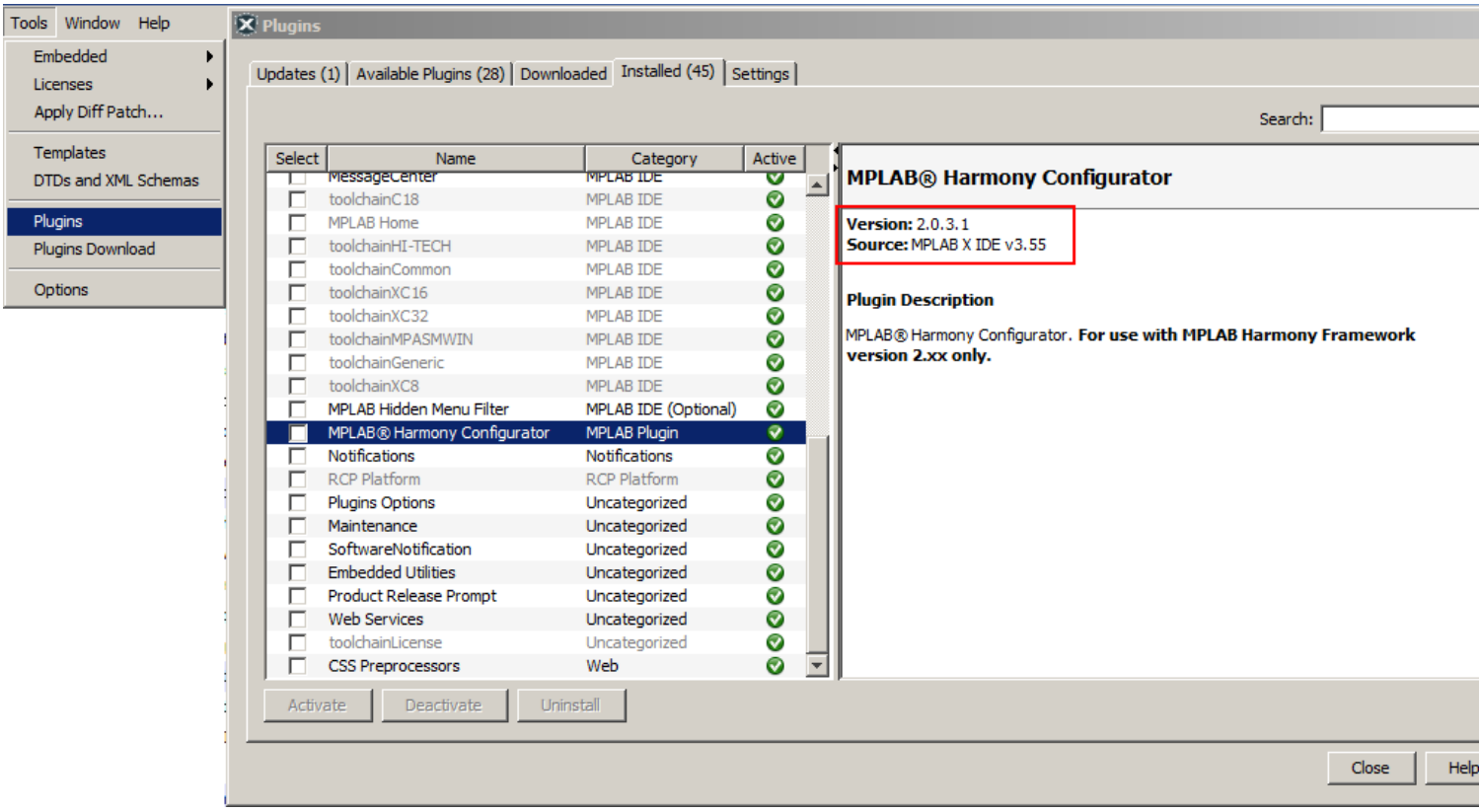
Provides information about MHC version numbering within MPLAB Harmony.

Description

Starting in version 2.02b, the architecture of the MPLAB Harmony Configurator has been separated into several sections and can be described, as shown in the following figure:



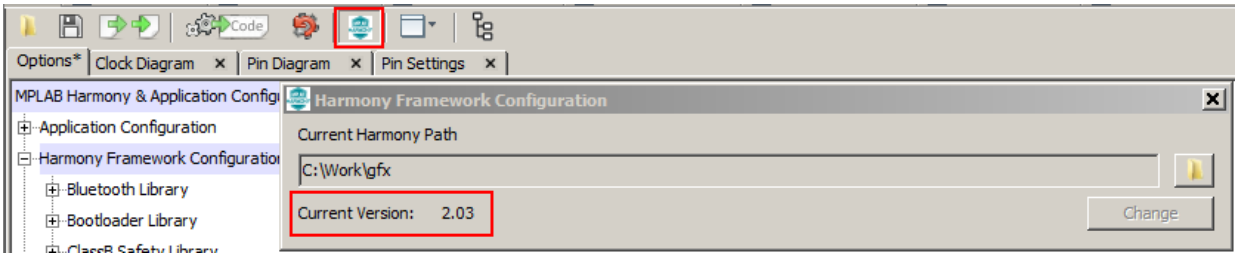
MPLAB X IDE Plug-in (MHC Loader) - This is the plug-in to MPLAB X IDE and is responsible for loading the MPLAB Harmony Configurator from the MPLAB Harmony Framework. Information about this plug-in can be obtained through MPLAB X IDE by navigating to *Tools* > *Plugins* > *Installed*.



MPLAB Harmony Configurator Core (mhc.jar) - This Java module contains the core code for the MHC system. Prior to version 2.02b, this functionality was contained with the MPLAB X IDE MHC Plug-in and would often cause the plug-in to be out of sync with a given MPLAB Harmony framework. In the v2.02b release, the core MHC module was moved to reside inside of the MPLAB Harmony framework itself to help decouple the MPLAB X IDE plug-in from the framework used. The core module is located in the `utilities/mhc` folder within the MPLAB Harmony framework installation.

Note: **MPLAB X IDE MHC Plug-ins v2.01 and earlier are not compatible with frameworks v2.02b and later and and vice versa**. While it is always recommended to keep the loader version and the framework version synchronized, from v2.02b forward, the MPLAB X IDE plug-in is expected to be backwards and forwards compatible between MHC core versions.

You can see the version of MPLAB Harmony that you are currently using by using the Framework Configuration dialog box inside MHC.



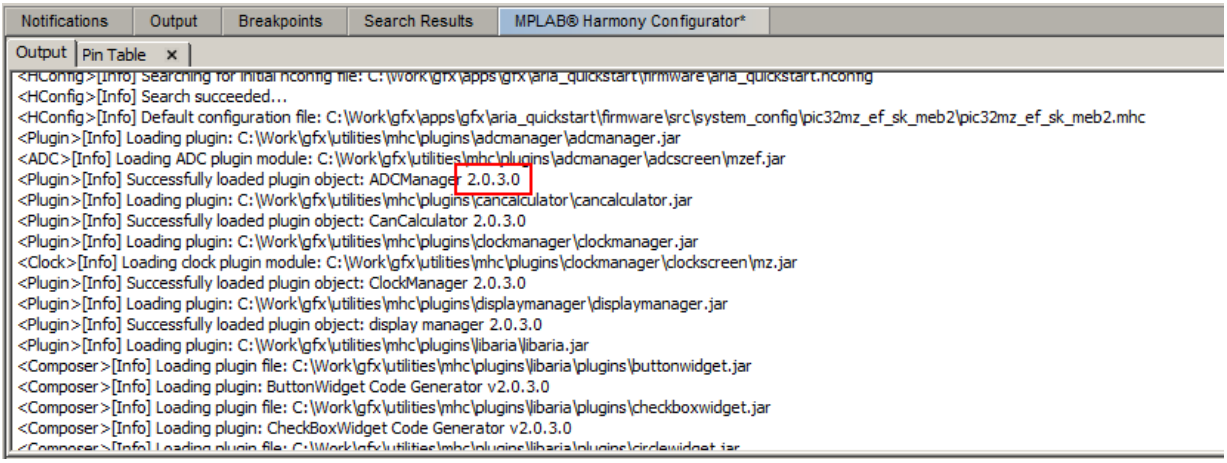
This dialog allows the user to change the MPLAB Harmony framework path that the application configuration is currently using. When opened, the dialog shows the current MPLAB Harmony path and the version number of the framework.

To change the framework that is used by the active configuration:

1. Open the Framework Configuration dialog box by click MPLAB Harmony icon.
2. Click **Browse** on the right side of the path text box.
3. Browse to the root path of the new framework for the configuration and click **OK**.
4. Click **Change** and accept the warning.
5. Save your project after making the change.

The application configuration will now use the new MPLAB Harmony framework.

MPLAB Harmony Configurator Plug-ins - MHC loads many plug-ins from the MPLAB Harmony framework. All of these plug-ins reside in `<install-dir>/utilities/mhc/plugins`. You can see the versions of these plug-ins in the Output Window when they are loaded by MHC.



Note: All plug-ins within a given MPLAB Harmony framework are guaranteed to be compatible with the version of MHC (`mhc.jar`) contained in that framework.

MPLAB Harmony Configurator Developer's Guide

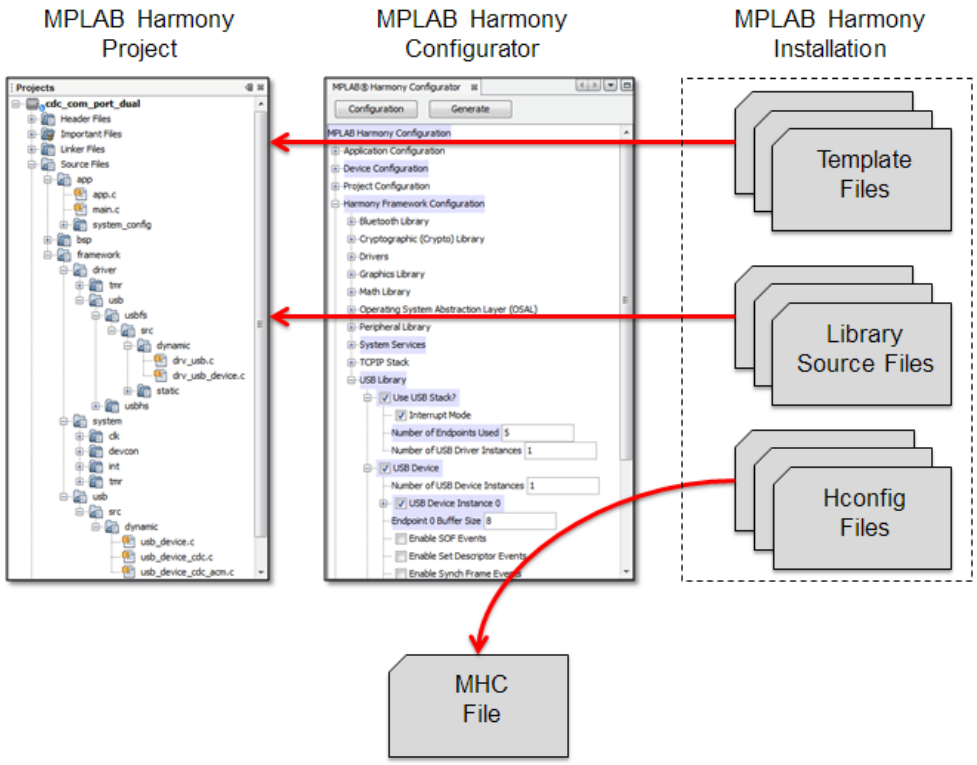
This section describes the basic operation of the MPLAB Harmony Configurator (MHC), the details of the hconfig, template, and `.mhc` files it utilizes, and explains how to add support for new libraries that are compatible with MPLAB Harmony into the MHC.

Introduction

This topic provides an introduction and overview of the MPLAB Harmony Configurator (MHC).

Description

When installed into MPLAB X IDE, the MHC plug-in provides a "New MPLAB Harmony" project wizard and a graphical user interface for configuration of MPLAB Harmony projects. When used, it generates (or updates) a project outline, including the C-language main function and system configuration files and stores the project configuration selections for later retrieval, modification, and sharing. To do this, the MHC utilizes a completely data driven method for defining the configuration options presented to the user and a template driven method for generating the source code, as illustrated in the following diagram.



Libraries are primarily provided in the MPLAB Harmony installation in source form. Each library provides an hconfig file and a set of template (`.ftl`) files. The hconfig files are text files that use an extended version of the Linux Kconfig grammar to define the configuration options available for the associated library and to identify source files, dependencies, and help content. When launched from within MPLAB X IDE, the MHC reads the hconfig files and presents the libraries and options to the user for selection and configuration in a graphical tree-based format similar to the Linux Xconfig utility.

After the user makes the desired library and configuration option selections and clicks **Generate**, the MHC stores the selections in another text file named for the current project configuration (as defined by the IDE) with an `.mhc` extension. Then, it processes the basic MPLAB Harmony template files, along with the templates for the selected libraries, using the Java FreeMarker engine to replace the markup text in the template files with the selections made by the user. It then generates the configuration-specific C-language source files for the current configuration of the current main project in the MPLAB X IDE. It also inserts the appropriate source (and/or binary) files for the selected libraries into the MPLAB X IDE project.

After the MHC generates the configuration, the resultant project will build and run, but it may not do anything useful until the user implements the desired application code.

Adding New Libraries

This section provides information on adding a new library to MPLAB Harmony.

Description

The process of adding a new library that is supported by MHC to a MPLAB Harmony installation consists of the following basic steps.

1. Develop a new MPLAB Harmony compatible library module.
2. Develop the hconfig file to define the library's configuration options and insert it into the MPLAB Harmony hconfig hierarchy.
3. Develop the FreeMarker templates to generate the necessary configuration-specific source code.
4. Insert the new FreeMarker templates into the MPLAB Harmony top-level templates.
5. Install the library source (and other supporting) files into the appropriate locations in the MPLAB Harmony installation tree.
6. Insert the library's documentation into the MPLAB Harmony documentation index.

These steps are described in detail in the following sections.

Developing a Library That is Compatible With MPLAB Harmony

This provides information on compatibility.

Description

MPLAB Harmony libraries need to be modular and inter-operable so that they can be configured for one or more of the different environments supported by MPLAB Harmony. To develop a library that is compatible with MPLAB Harmony, it must meet the design and implementation guidelines as described in the MPLAB Harmony Compatibility Guide. Please refer to this section for the modularity, flexibility, testing, and documentation guidelines that are required and recommended for MPLAB Harmony. Please ensure that any library added to the MPLAB Harmony framework meets these guidelines.

Developing a New hconfig File

This topic lists and describes the steps necessary when developing a new hconfig file.

Description

The configuration options for a library are wholly defined within the hconfig file(s) associated with that library. This section describes how to create an hconfig file for a new driver module. The process is as follows:

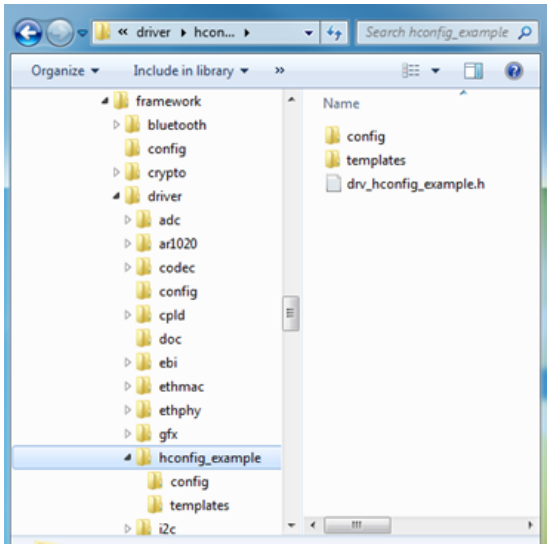
- Step 1: Create the File and Insert it into the hconfig Hierarchy
- Step 2: Create a Menu Item for the Module in the Driver Framework Tree
- Step 3: Creating Configuration Options
- Step 4: Use Dependencies
- Step 5: Use the Choice and Select Statements to Enable One Module Needed by Another
- Step 6: Sourcing hconfig Files
- Step 7: Adding Source Files to the MPLAB X IDE Project With the "file" Statement
- Step 8: Add Help Links to Configuration Options
- Step 9: Create Multiple Module Instances

Step 1: Create the File and Insert it into the hconfig Hierarchy

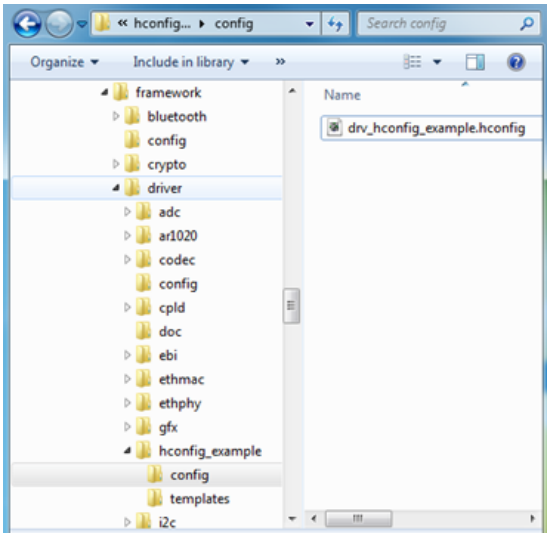
This topic describes how to create the hconfig file and insert it into the hconfig hierarchy.

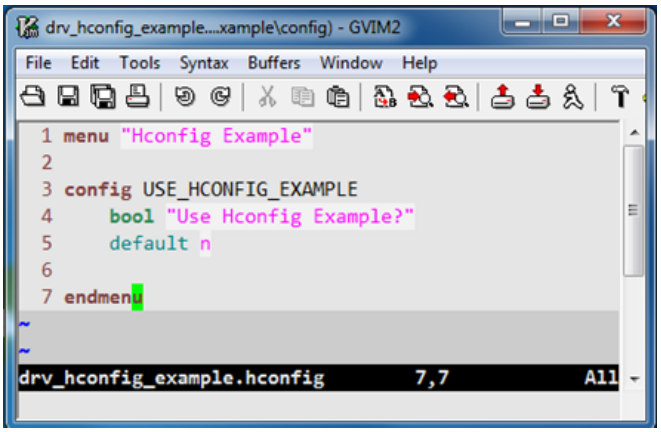
Description

Our module example will be a MPLAB Harmony driver named "hconfig_example", and will be inserted into the `<$HARMONY_VERSION_PATH>/framework/driver` directory.



Let's create an hconfig file for the hconfig_example driver, and put it in the `config` folder. For now, all it will do is create a menu entry and a single Boolean config item "Use Hconfig Example?". Note that by default, the driver is not selected.





Now we need to insert our hconfig file into the hconfig tree hierarchy so it will be invoked when we run MHC.

Driver hconfig files are sourced from the `<$HARMONY_VERSION_PATH>/framework/driver/config/driver.hconfig` file.

```
menu "Drivers"

config DRIVER
    bool

source "$HARMONY_VERSION_PATH/framework/driver/ar1020/config/drv_ar1020.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/ebi/config/drv_ebi.hconfig"
#source "$HARMONY_VERSION_PATH/framework/driver/ethmac/config/drv_ethmac.hconfig"
#source "$HARMONY_VERSION_PATH/framework/driver/ethphy/config/drv_ethphy.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/gfx/config/drv_gfx.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_example.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/i2s/config/drv_i2s.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/ic/config/drv_ic.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/nvm/config/drv_nvm.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/oc/config/drv_oc.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/pmp/config/drv_pmp.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/rtcc/config/drv_rtcc.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/sdcard/config/drv_sdcard.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/spi/config/drv_spi.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/tmr/config/drv_tmr.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/usart/config/drv_usart.hconfig"
#source "$HARMONY_VERSION_PATH/framework/driver/usb/config/drv_usb.hconfig"
#source "$HARMONY_VERSION_PATH/framework/driver/wifi/config/drv_wifi.hconfig"
endmenu

ifblock DRIVER
file DRIVER_H "$HARMONY_VERSION_PATH/framework/driver/driver.h"
endif
```

Added

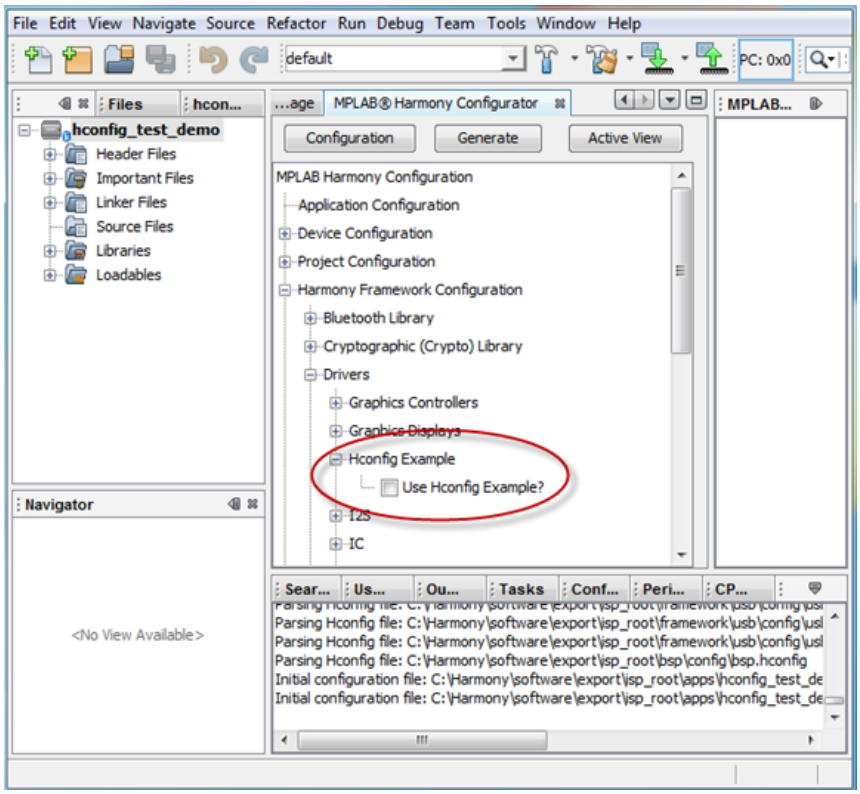
Note that all hconfig files are included recursively by the top-level hconfig file in the application's firmware directory. The entire hconfig tree is parsed when MHC is invoked and when a configuration change is made, so the relative placement of configuration options only affects the menu structure. There is no functional dependency.

Step 2: Create a Menu Item for the Module in the Driver Framework Tree

This topic describes creating a menu item for the MPLAB Harmony module in the Driver framework tree.

Description

Let's create a demonstration application and see if our driver config appears.



Step 3: Creating Configuration Options

This topic describes adding menu configuration options

Description

Now let's add some config options.

- A config option selected from a drop-down menu
- A Boolean config value
- An integer config whose default value depends on the first config option

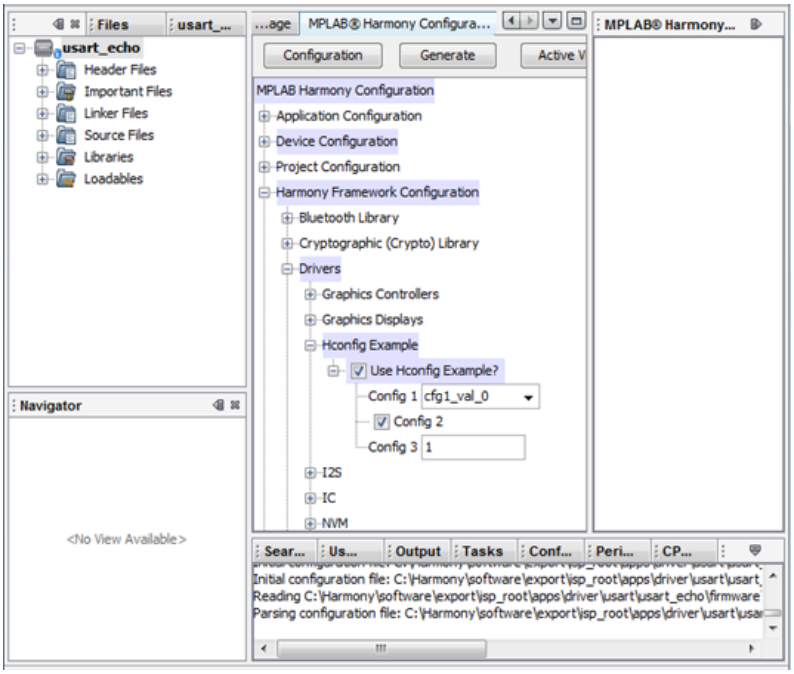
```
menu "Hconfig Example"

config USE_HCONFIG_EXAMPLE
    bool "Use Hconfig Example?"
    default n

# 1. A config option selected from a dropdown menu.
enum CFG1_VAL
    "cfg1_val_0"
    || "cfg1_val_1"
    || "cfg1_val_2"

config CFG1
    depends on USE_HCONFIG_EXAMPLE
    string "Config 1"
    range CFG1_VAL
    default "cfg1_val_0"

# 2. A boolean config value
config CFG2
    depends on USE_HCONFIG_EXAMPLE
    bool "Config 2"
    default y

# 3. An integer config whose default value depends on CFG1.
config CFG3
    depends on USE_HCONFIG_EXAMPLE
    int "Config 3"
    range 1 3
    default 1 if CFG1 = "cfg1_val_0"
    default 2 if CFG1 = "cfg1_val_1"
    default 3 if CFG1 = "cfg1_val_2"

endmenu
```

Step 4: Use Dependencies

This topic describes use dependencies.

Description

Note that all config options have a dependency on USE_HCONFIG_EXAMPLE. This means that they will not be visible in the MHC menu unless USE_HCONFIG_EXAMPLE is true. Also note the range on CFG3. An attempt to set CFG3 to a value outside the listed range will be flagged as an error in MHC.



The default value of "Config 3" is set according to Config 1. The first true default in a config block becomes the default value of the config option, and any subsequent default statements are ignored. Therefore, if we add a default with no `if` clause ahead of the other defaults, it will become the default of the config option regardless of whether or not any of the others are true.

A config option may contain multiple dependencies. Both dependencies and `if` statements can contain logical `AND` and `OR`.

```
33 config CFG4
34     depends on USE_HCONFIG_EXAMPLE && CFG2
35     bool "Config 4"
36     default n
37
38 config CFG5
39     depends on USE_HCONFIG_EXAMPLE && CFG2
40     depends on CFG4
41     bool "Config 5"
42     default y if (CFG1 = "cfg1_val_0") && CFG4
43
```

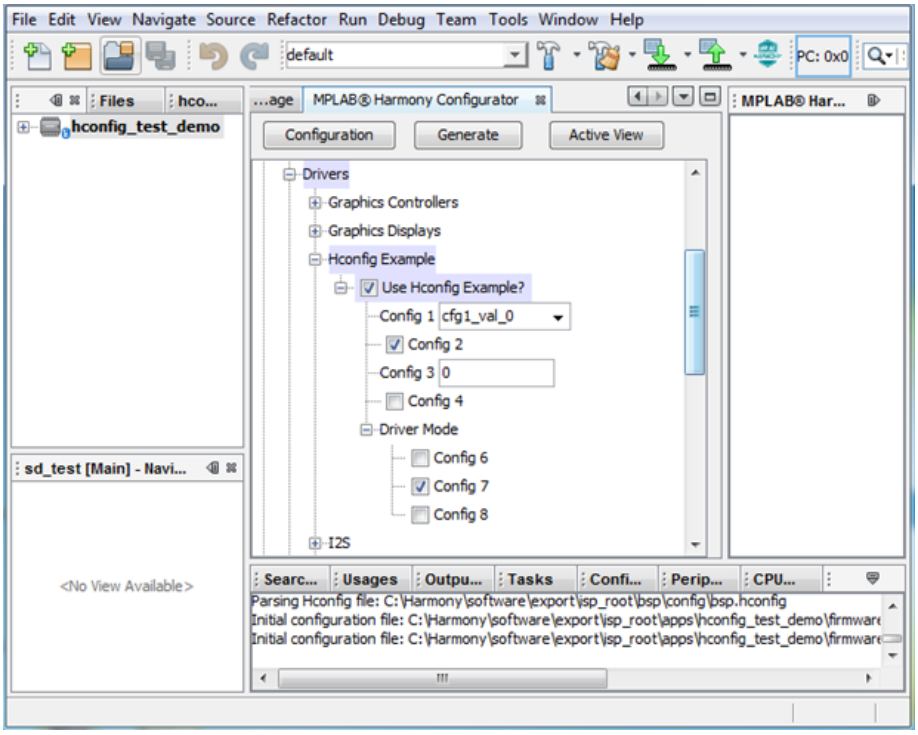
Step 5: Use the Choice and Select Statements to Enable One Module Needed by Another

This topic describes using the "choice" and "select" statements to enable a module to be used by another module.

Description

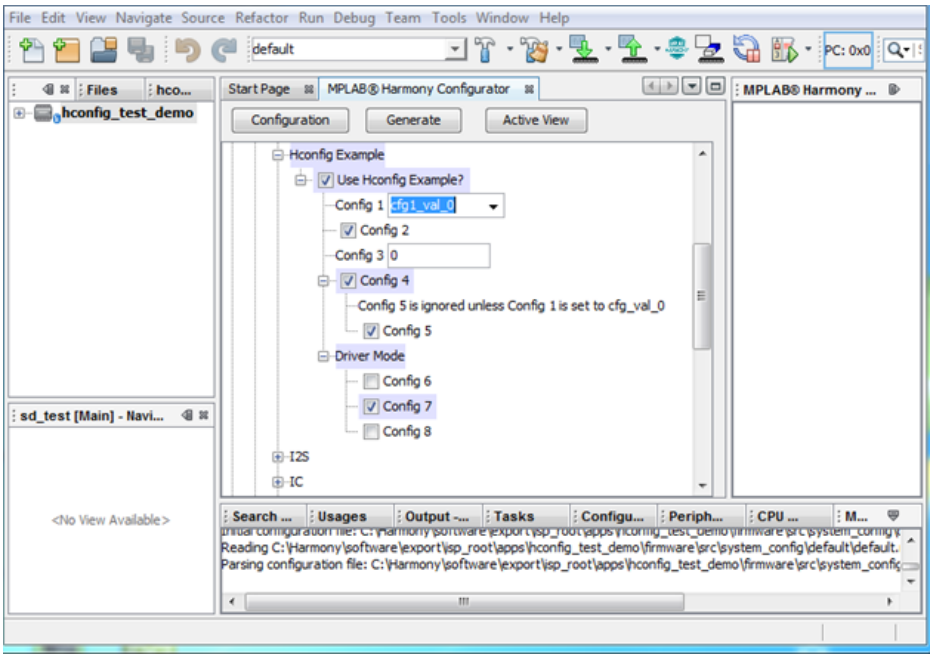
You can make config options mutually exclusive with the "choice" statement. This is useful for modules that can be configured to operate in different modes. A choice block requires a prompt, which is displayed in the hconfig tree. Choice blocks can optionally have a default option and dependencies. If no default is provided, the choice block will be flagged in red until one of the config options is checked.

```
45 choice
46     prompt "Driver Mode"
47     depends on USE_HCONFIG_EXAMPLE
48     default CFG7
49 config CFG6
50     bool "Config 6"
51
52 config CFG7
53     bool "Config 7"
54
55 config CFG8
56     bool "Config 8"
57 endchoice
```



Comments can be displayed in the menu with the "comment" statement. Comments can also have dependencies.

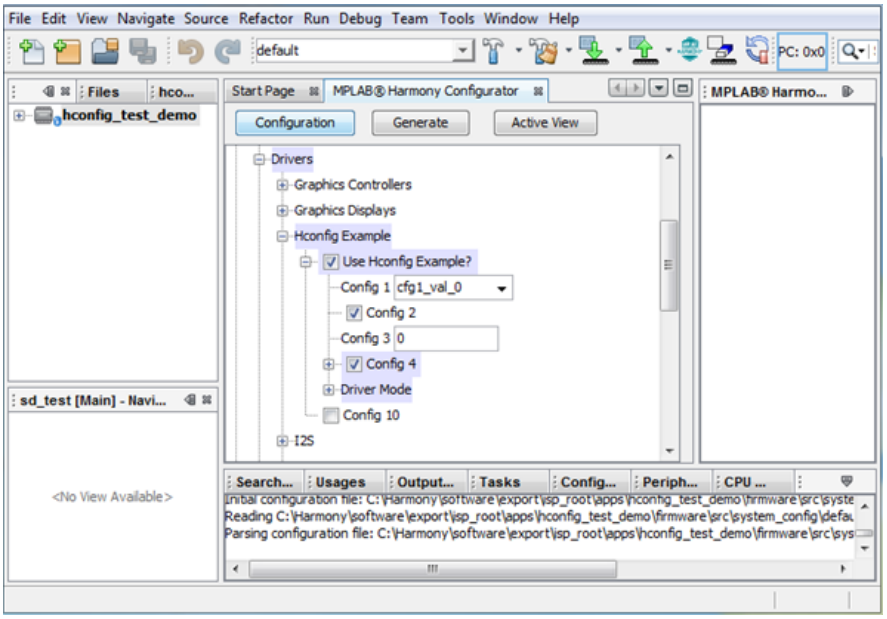
```
39 comment "Config 5 is ignored unless Config 1 is set to cfg_val_0"
40     depends on CFG4
41 config CFG5
42     depends on USE_HCONFIG_EXAMPLE && CFG2
43     depends on CFG4
44     bool "Config 5"
45     default y if (CFG1 = "cfg1_val_0") && CFG4
46
```

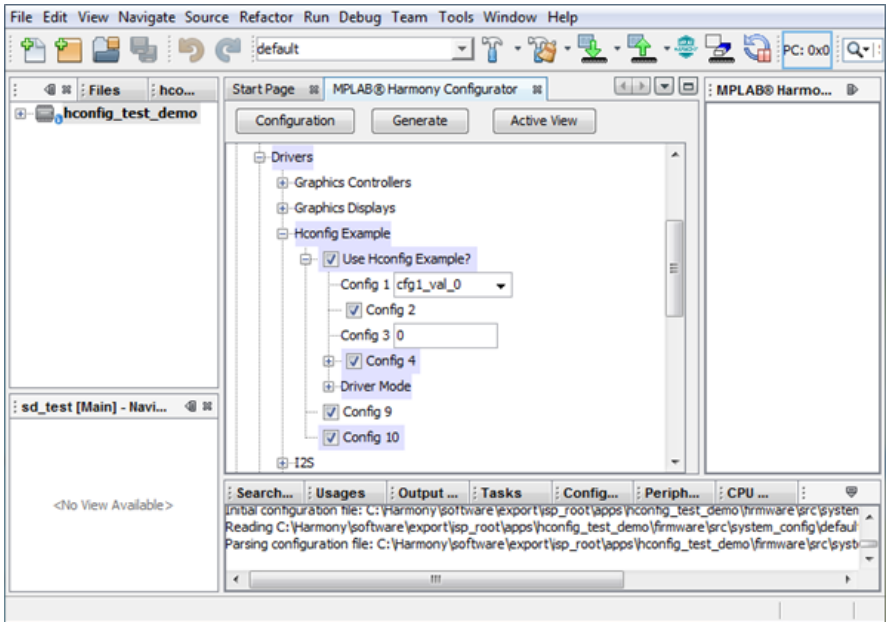


The "select" statement is used to select or enable a config option based on another config option. This is often used to enable a module that may be required by multiple modules. An example is the Interrupt System Service, which is used by many drivers and system services.

The select statement must be part of a config block. It should only be used to select non-visible config options. The reason for this is that once a config option is selected, it cannot be unselected. Even if the config option is not checked in the menu, it will still be selected in hconfig, and included in the generated code.

```
config CFG9_NEEDED
    bool

config CFG9
    depends on CFG9_NEEDED
    bool "Config 9"
    default y if CFG9_NEEDED
    default n

config CFG10
    bool "Config 10"
    default n
    select CFG9_NEEDED
```





Step 6: Sourcing hconfig Files

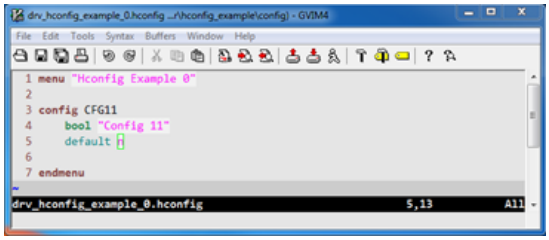
This topic describes how to source an hconfig file from another hconfig file.

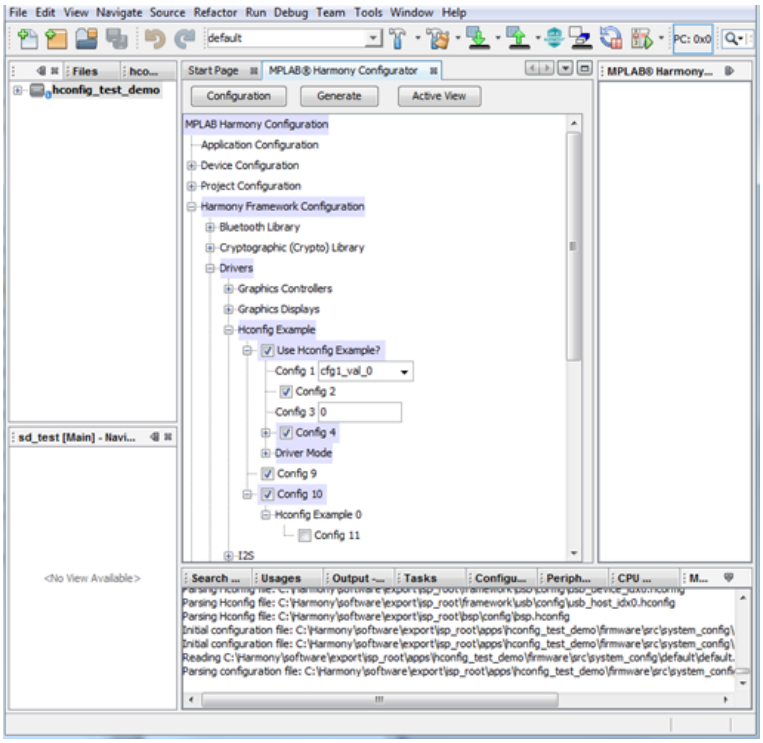
Description

An hconfig file can source other hconfig files. This is useful for grouping related config options or handling multiple module instances. The sourced file may optionally contain a menu/endmenu block.

Enclosing a source statement within an "ifblock" will apply the dependency to all config options within the sourced file. In the example shown below, all config options in the `drv_hconfig_example_0.hconfig` file are dependent on CFG10. All Ifblock statements must be terminated with `endif`.

```
74 ifblock CFG10
75 source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_example_0.hconfig"
76 endif
77
```





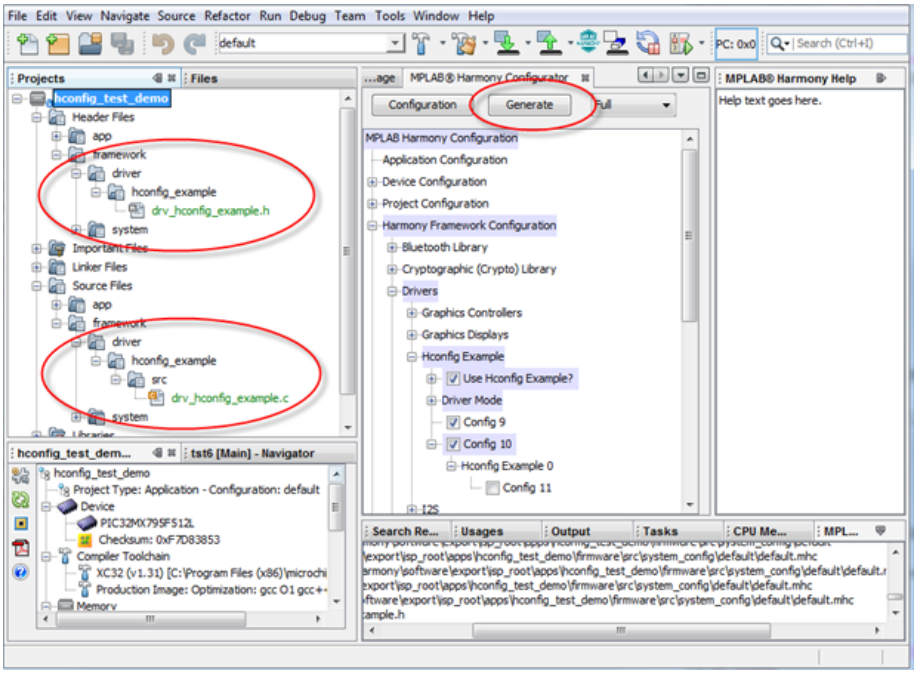
Step 7: Adding Source Files to the MPLAB X IDE Project With the "file" Statement

This topic describes adding source files using the "file" statement.

Description

MHC adds source files to the MPLAB X IDE project with the "file" statement. The full path to the file on disk must be provided, as well as the virtual directory in MPLAB X IDE. The "file" statement does not copy files, it just adds existing files to the MPLAB X IDE project. The files are added when the user clicks **Generate** within MHC.

```
82
83 source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_idx.ftl" 3 instances
84
85 file DRV_HCONFIG_EXAMPLE_H "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/drv_hconfig_example.h" to
  "$PROJECT_HEADER_FILES/framework/driver/hconfig_example/drv_hconfig_example.h"
86 file DRV_HCONFIG_EXAMPLE_C "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/src/drv_hconfig_example.c"
   to "$PROJECT_SOURCE_FILES/framework/driver/hconfig_example/src/drv_hconfig_example.c"
87
```



Step 8: Add Help Links to Configuration Options

This topic provides an example for adding Help links.

Description

Each configuration option may have Help text associated with it. In MHC, the Help text is a hyperlink into the MPLAB Harmony documentation. If no link exists, the text itself is displayed in the help window.

```
69 config CFG10
70     bool "Config 10"
71     default n
72     select CFG9_NEEDED
73     ---help---
74     Help text goes here.
75     ---endhelp---
76
```

Step 9: Create Multiple Module Instances

This topic describes the creation of multiple instances.

Description

Several modules support multiple instances, requiring separate configuration options for each instance. In this case, the configuration options of different instances are identical, but may be set to different values. This is handled in MHC by a combination of the "instances" keyword, and a FreeMarker template that is processed once for each instance of the module. As an example, we will create three instances of our hconfig demonstration driver, each containing two configuration options.

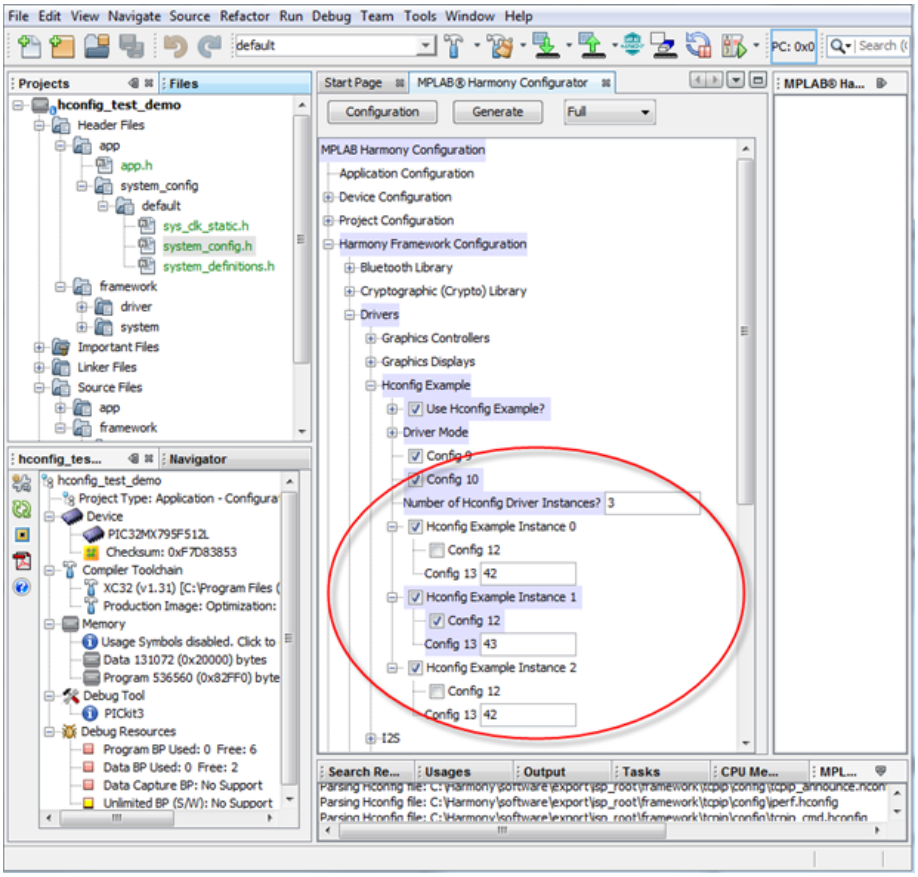
The instance template is sourced like a normal hconfig file, but with the keyword "instances" preceded by the maximum number of instances supported. A configuration option is added to allow the user to select the number of instances actually configured and instantiated.

```
77 config DRV_HCONFIG_INSTANCES_NUMBER
78     depends on USE_HCONFIG_EXAMPLE
79     int "Number of Hconfig Driver Instances?"
80     default 1
81     range 1 3
82
83 source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_idx.ftl" 3 instances
84
```

The FreeMarker template is a marked-up hconfig file that is processed through FreeMarker once for each instance. Each time it is processed, the `${INSTANCE}` variable is set to the instance number.

```
config DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE+1}
    bool
<#if INSTANCE != 0>
    default n if DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE} = n
</#if>
    default n if DRV_HCONFIG_INSTANCES_NUMBER = ${INSTANCE+1}
    default y

config DRV_HCONFIG_INST_IDX${INSTANCE}
    depends on USE_HCONFIG_EXAMPLE
<#if INSTANCE != 0>
                && DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE}
</#if>
    bool "Hconfig Example Instance ${INSTANCE}"
    default y

ifblock DRV_HCONFIG_CFG12_IDX${INSTANCE}

config DRV_HCONFIG_IDX${INSTANCE}
    bool "Config 12"
    default n

config DRV_HCONFIG_CFG13_IDX${INSTANCE}
    int "Config 13"
    default 42

endif
```

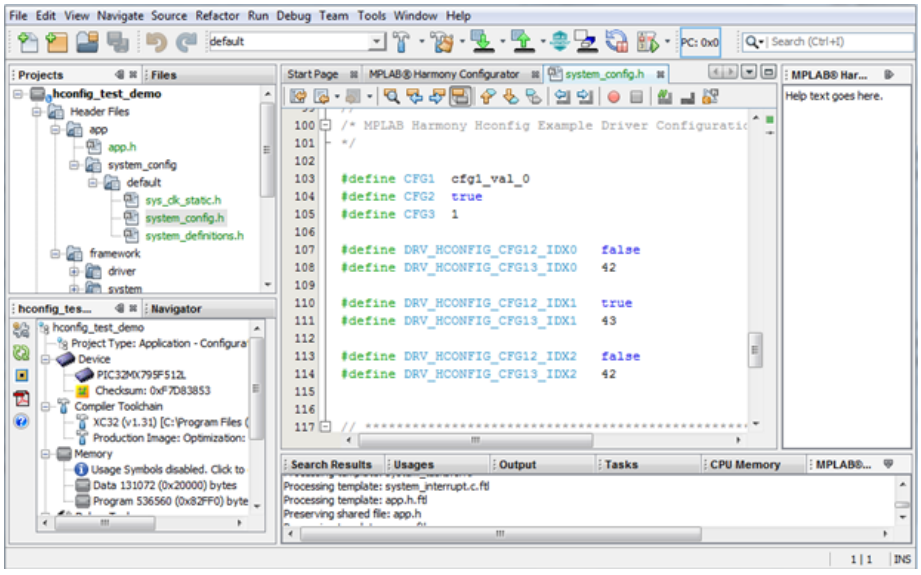
When MHC is run, the user is prompted for the number of instances. Configuration options for each instance are displayed in the menu.



The FreeMarker templates that are used to generate code must also be updated for multiple instances.

```
<#if CONFIG_USE_HCONFIG_EXAMPLE == true>

// *****************************************************************************
/* MPLAB Harmony Hconfig Example Driver Configuration Options
*/

#define CFG1  ${CONFIG_CFG1}
<#if CONFIG_CFG2 == true>
#define CFG2  true
<#else>
#define CFG2  false
</#if>
#define CFG3  ${CONFIG_CFG3}

<#-- Instance 0 -->
<#if CONFIG_DRV_HCONFIG_INST_IDX0 == true>
<#if CONFIG_DRV_HCONFIG_CFG12_IDX0 == true>
#define DRV_HCONFIG_CFG12_IDX0   true
<#else>
#define DRV_HCONFIG_CFG12_IDX0   false
</#if>
#define DRV_HCONFIG_CFG13_IDX0   ${CONFIG_DRV_HCONFIG_CFG13_IDX0}
</#if>

<#-- Instance 1 -->
<#if CONFIG_DRV_HCONFIG_INST_IDX1 == true>
<#if CONFIG_DRV_HCONFIG_CFG12_IDX1 == true>
#define DRV_HCONFIG_CFG12_IDX1   true
<#else>
#define DRV_HCONFIG_CFG12_IDX1   false
</#if>
#define DRV_HCONFIG_CFG13_IDX1   ${CONFIG_DRV_HCONFIG_CFG13_IDX1}
</#if>

<#-- Instance 2 -->
<#if CONFIG_DRV_HCONFIG_INST_IDX2 == true>
<#if CONFIG_DRV_HCONFIG_CFG12_IDX2 == true>
#define DRV_HCONFIG_CFG12_IDX2   true
<#else>
#define DRV_HCONFIG_CFG12_IDX2   false
</#if>
#define DRV_HCONFIG_CFG13_IDX2   ${CONFIG_DRV_HCONFIG_CFG13_IDX2}
</#if>

</#if>
```

When the code is generated, code is generated for each instance.



Using the Set Statement

Demonstrates how to use the `set` statement to configure dependencies.

Description

Often one MPLAB Harmony library uses (depends upon) another and has specific requirements on how that library must be configured. To illustrate this, the following Hconfig code MHC Options menu items to allow selection and configuration of a library (library C) that might be shared by other libraries.

Library C Selection and Configuration Menu Definition

```
# Library C Configuration
config USE_LIBRARY_C
    bool "Use Library C?"
    default n

menu "Configure Library C"
    depends on USE_LIBRARY_C

config LIBRARY_C_ITEM_1
    depends on USE_LIBRARY_C
    int "Library C, Item 1: Enter an integer"
    default 0

endmenu  # Configure Library C
```

If another library (library A) requires the use of library C and requires library C's configuration item (LIBRARY_C_ITEM_1) to have a specific value (42), the following Hconfig code will define an MHC options menu to satisfy this requirement.

Library A Selection and Configuration Menu Definition

```
# Library A Configuration
config USE_LIBRARY_A
    bool "Use Library A?"
    default n
    set USE_LIBRARY_C to y if USE_LIBRARY_A = y
    set LIBRARY_C_ITEM_1 to 42 if USE_LIBRARY_A = y

comment "Sets Library C, Item 1 to 42"
    depends on USE_LIBRARY_A

menu "Configure Library A"
    depends on USE_LIBRARY_A

config LIBRARY_A_ITEM_1
    depends on USE_LIBRARY_A
    int "Library A, Item 1: Enter an integer"
    default 0

endmenu  # Configure Library A
```

However, if a second library (library B) also depends on library C, it is possible that the default configuration settings for library C that it requires may be different. This is shown in the following Hconfig code that defines library B's selection and configuration menu, and uses the `set` statement to set library C's item 1 to a value of 86.

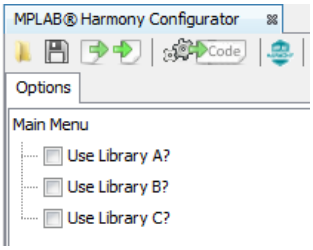
Library B Selection and Configuration Menu Definition

```
# Library B Configuration
config USE_LIBRARY_B
    bool "Use Library B?"
    default n
    set USE_LIBRARY_C to y if USE_LIBRARY_B = y
    set LIBRARY_C_ITEM_1 to 86 if USE_LIBRARY_B = y

comment "Sets Library C, Item 1 to 86"
    depends on USE_LIBRARY_B

menu "Configure Library B"
    depends on USE_LIBRARY_B

config LIBRARY_B_ITEM_1
    depends on USE_LIBRARY_B
    int "Library B, Item 1: Enter an integer"
```

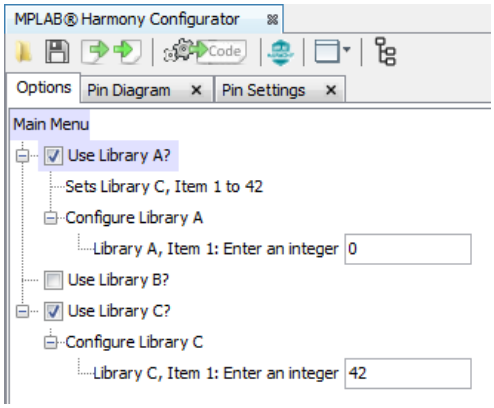
```
    default 0

endmenu  # Configure Library B
```

When such a conflict occurs, the MHC notifies the user, who is then required to enter a value to resolve the conflict (if possible) or disable one of the dependent libraries.

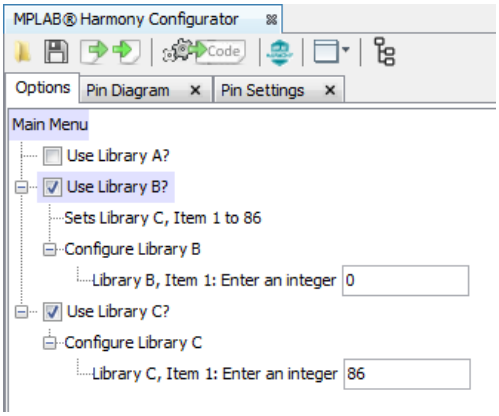
The following sequence of images illustrates the behavior of the MHC when the previous Hconfig code is used. Before any of these libraries have been selected, the MHC Options menu shows their Use Library options.



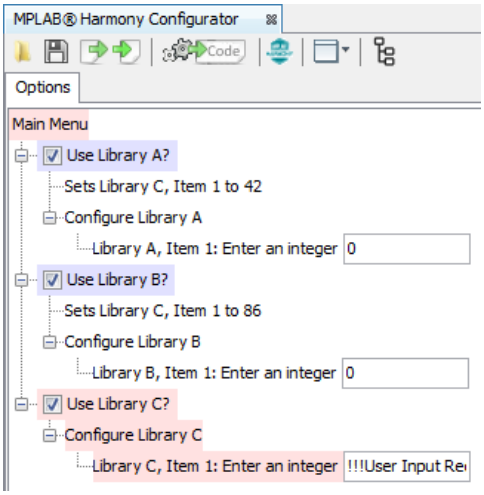
If library A is used (but not library B), the MHC automatically sets the value of library C's configuration item to 42.



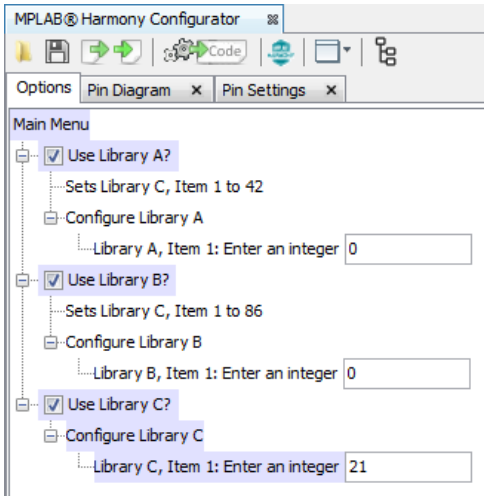
If library B is used (but not library A), the MHC automatically sets the value of library C's configuration item to 86.



However, if both library A and B are used, the MHC highlights the conflict in library C in red and requires the user to enter a value to resolve the conflict.



If the user then enters a value for library C's item 1, the MHC recognizes that the user has set that item's value and assumes that the conflict has been resolved.



It is important to understand that the MHC does not validate that the chosen value satisfies the requirements of both libraries A and B. It is up to the user to understand the requirements and select an appropriate configuration value.

It is also a good practice to provide a comment in the dependent library's configuration menu when it sets dependencies so that the user knows it has done so.

hconfig Development Guidelines

This topic describes the conventions and guidelines to be used when creating hconfig files.

Description

The following conventions need to be followed when developing MPLAB Harmony hconfig files:

- HAVE_<peripheral> configuration options are used to indicate whether or not a specific peripheral is supported on the device. These options are non-visible, Boolean, and primarily located in `framework.hconfig`. They are set to 'y' using the "select" keyword in the processor-specific peripheral hconfig files. The processor-specific peripheral hconfig files are generated automatically from processor-specific PLIB header files.
- All hconfig files shall be placed in a "config" folder in the MPLAB Harmony framework tree. The hconfig files shall "source" other hconfig files lower in the framework hierarchy. For example the framework hconfig file sources an hconfig file for each folder in the framework directory. The driver hconfig file sources an hconfig file for every driver in the framework/driver directory, and so on.
- The keyword "select" shall not be used with visible config options. Once something is selected using the "select" keyword, it is always selected, regardless of whether or not it is checked in the MHC menu.
- When sourcing an hconfig file within an ifblock, the file is always sourced, and the ifblock dependencies are applied to all items within the sourced file
- Adding the keyword "exclusive" to an enum definition prevents the same element from being assigned to more than one config option
- There can be only one "mainmenu". The top-level hconfig file containing the mainmenu is generated by MHC and placed in the application firmware directory. The template for the top-level hconfig file is located in `utilities/mhc/config`.
- It is often useful to have modules enable each other. The mechanism for this is to use the "select" keyword within one module to select a non-visible config option within another module. The non-visible config option is then used as a dependency for the first module. By convention, the non-visible option is named `USE_<module>_NEEDED`. For example, the Timer System Service requires a Timer Driver instance. The Timer

Driver hconfig contains:

```
config USE_DRV_TMR_NEEDED
    bool
config USE_DRV_TMR
    depends on HAVE_TMR
    bool "Use Timer Driver?"
    default y if USE_DRV_TMR_NEEDED
    default n
```

and the timer system service hconfig contains:

```
config USE_SYS_TMR
    bool "Use Timer System Service?"
    select USE_DRV_TMR_NEEDED
    default y if USE_SYS_TMR_NEEDED
    default n
```

Selecting the Timer System Service automatically selects the Timer Driver, by selecting `USE_DRV_TMR_NEEDED`, which (if selected) sets the `USE_DRV_TMR` default to 'y'.

- When multiple default values are given to a config option, the first one that evaluates to *true* becomes the config option value
- By convention, the selection of a module is made in the menu with the menu text "Use <module>?" (e.g., "Use Timer Driver?")
- Modules should default to not-used unless selected by another module
- Visible config options should follow MPLAB Harmony naming conventions
- When selecting module features, menu entries should include a feature rather than exclude, and enable rather than disable. Default for all visible config options should be excluded or disabled, unless needed by another module or enabled by dependencies.
- All visible config options must have an associated help tag, and must be documented in the Help documentation
- Visible config options and comments must capitalize each word in menu text
- Integer config options should have a range whenever possible

Developing MPLAB Harmony FreeMarker Templates

This topic provides information on developing FreeMarker templates.

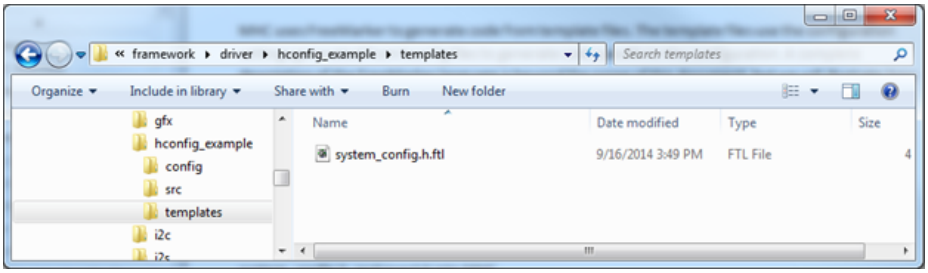
Description

MHC uses FreeMarker to generate code from template files. The template files use the configuration settings generated from hconfig files to generate code specific to the configuration. A complete description of the FreeMarker language is beyond the scope of this document. Please refer to the online FreeMarker manual, which available at: http://freemarker.org/docs/. This section will illustrate how MHC uses it with a simple example.

The configuration options generated by MHC are written to a `<configuration>.mhc` file in the project's `firmware/src/system_config/<configuration>` directory. In our example, the project name is `hconfig_test_demo`, and the configuration is "default". By default, a number of files are generated by MHC and placed in the application's `firmware/src` directory. The configuration-specific files are in the `firmware/src/ system_config/<configuration>` directory. The configuration options are written to the system_config.h file. For this example, we will first show how to create a FreeMarker template for our `system_config.h`, and insert it into MHC.

For this example, we will use a simple version of `drv_hconfig_example.hconfig`, with just three config options, CFG1, CFG2, and CFG3.

First, we need to create the template for `system_config.h`. By convention, this file will be named `system_config.h.ftl`, and be placed in the `<module>/templates` directory.



Second, we need to implement the source code template, as shown by the following example.

```
42 *******************************************************************
43 -->
44 <#if CONFIG_USE_HCONFIG_EXAMPLE == true>
45
46 // *******************************************************************
47 /* MPLAB Harmony Hconfig Example Driver Configuration Options
48 */
49
50 #define CFG1  ${CONFIG_CFG1}
51 <#if CONFIG_CFG2 == true>
52 #define CFG2  true
53 <#else>
54 #define CFG2  false
55 </#if>
56 #define CFG3  ${CONFIG_CFG3}
57
58 </#if>
59 <#--
60 /*******************************************************************
```

The source code template will include FreeMarker "markup" statements (defined between the <# and > escape tags and will use FreeMarker variables (defined between the ${ and } escape tags. The FreeMarker statements are interpreted semantically by the FreeMarker engine and the variables are textually replaced using values defined using the MHC by the user and stored in the `.mhc` file.

Note: Symbols defined in hconfig files must be prefixed with `CONFIG_` to use them in FreeMarker templates.

The resulting customized source code is generated directly into the configuration-specific folder of the current project.

Device Configuration

This topic describes the CONFIG_DEVICE hconfig symbol.

Description

CONFIG_DEVICE

This hconfig symbol can be used to provide the device ID based on the device selected in MPLAB X IDE. This feature is useful if hconfig/FTL logic that is unique to a device variant needs to be added.

The following example shows the FTL code to perform a check for a specific device:

```
<#if CONFIG_DEVICE == "PIC32MZ2028ECM144">
  … perform device-specific code …
<#/if>
```

Insert the New FreeMarker Templates into the MPLAB Harmony Top-level Templates

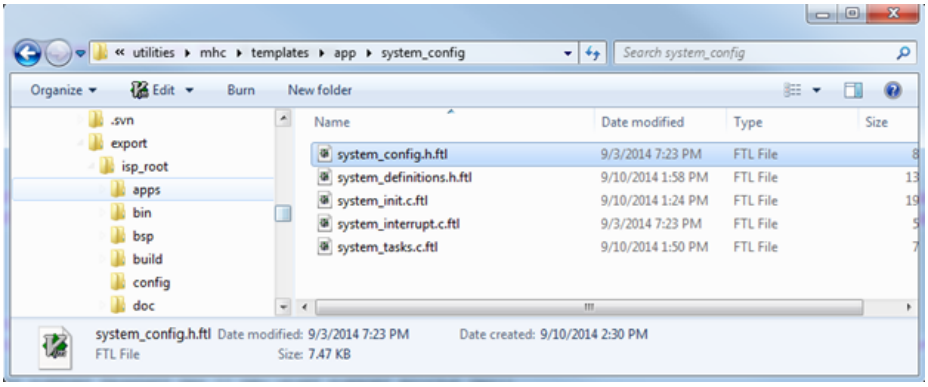
This topic describes how to insert a new FreeMarker template into the MPLAB Harmony top-level templates.

Description

To insert the template into MHC, we need to do one of two things:

- Use the "template" keyword in hconfig, or
- Include this template into another template

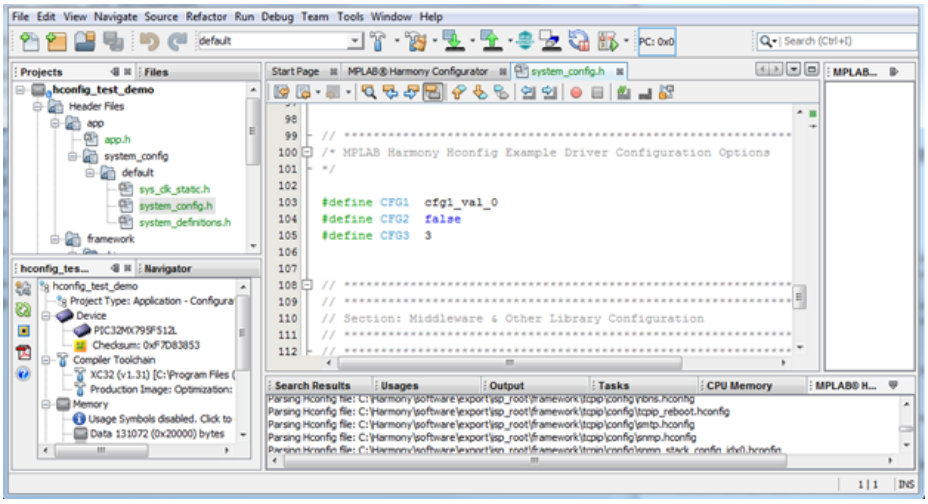
Since the `system_config.h` file draws config options from many templates, we will include our template in the top-level `system_config.ftl`. It is located in the `$HARMONY_VERSION_PATH/utilities/mhc/templates/app/system_config` directory.



This top-level template simply includes all of the module-specific templates that contribute to the `system_config.h` file. The included files are logically organized within the top-level template. For the following example, we will add our template in the driver configuration section.

```
118 // *****************************************************************************
119 // *****************************************************************************
120 // Section: Driver Configuration
121 // *****************************************************************************
122 // *****************************************************************************
123
124 <#if CONFIG_USE_DRV_TMR == true>
125 <#include "/framework/driver/tmr/templates/drv_tmr.h.ftl">
126 </#if>
127 <#if CONFIG_USE_DRV_USART == true>
128 <#include "/framework/driver/usart/templates/drv_usart.h.ftl">
129 </#if>
130 <#if CONFIG_USE_HCONFIG_EXAMPLE == true>
131 <#include "/framework/driver/hconfig_example/templates/system_config.h.ftl">
132 </#if>
```

When we generate the code, we see our config options are now in `system_config.h`.



Code generation for the rest of the system files follows the same process. A library typically needs to insert code into the following template system configuration template files:

`system_config.h.ftl`	Configuration item definitions
`system_definitions.h.ftl`	Configuration data types, object handles, and include statements
`system_init.c.ftl`	Init data structure definition and call to initialize function
`system_interrupt.c.ftl`	Raw ISR and call to tasks function, if interrupt driven
`system_tasks.c.ftl`	Call to tasks function, if polled

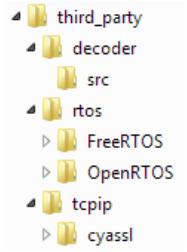
It will be necessary to modify each of the above templates to include the module-specific templates for any new libraries. It is also necessary to carefully review each top-level template to determine the appropriate location at which to include the module-specific templates and then test the code that is generated to ensure that it does not contain any FreeMarker engine error messages and that it functions as expected.

Installing a New Library into MPLAB Harmony

This topic provides information on inserting a new library into MPLAB Harmony.

Description

Within the MPLAB Harmony installation, you will find a `third_party` top-level folder, as shown in the follow figure. Within that folder, third-party code is organized by its purpose. If an appropriate sub-folder exists, create a directory named for your company or your product within that folder and copy your source installation files and folders into it. Your source tree should include the necessary hconfig and FreeMarker templates (as described previously) and Help content (described in the next section) to support your library in the MHC.



By default, MPLAB Harmony installs into a version-specific folder (`C:\microchip\harmony\<version>` on a Windows personal computer or `~/microchip/harmony/<version>` on a Mac or Linux computer). Therefore, when you install a newer version of MPLAB Harmony, it is very likely that you will need to reinstall your library. If your library is not part of the MPLAB Harmony installation, providing an installer that automates the process of copying your installation files into the new MPLAB Harmony installation folder and inserting the hconfig, FreeMarker templates, and help files into the new hierarchies will be a necessity.

Inserting New Library Help into the MPLAB Harmony Documentation Index

This topic provides information on inserting Help created for a new library into the existing MPLAB Harmony Help.

Description

The MHC displays Help information for each option when it is selected (i.e., clicked) by the user in the configuration window. To do this, the MHC reads the first word (token of contiguous characters with no whitespace) in the Help (or "---help---") section in the associated hconfig file. This word is assumed to be an index entry in the `install-dir>/doc/html/help_harmony_html_alias.h` header file in the selected MPLAB Harmony installation. If the MHC finds this entry in the alias file, it opens the associated HTML file in the Help window pane. If it does not find this entry in the alias header file, it displays the actual text provided in the Help section of the hconfig file. Therefore, there are two ways to support Help documentation in the MHC.

HTML Browser Used by MHC

This topic provides information on the HTML browser used by the MHC to display Help content.

Description

The HTML browser used by MHC is the GUI widget, HTMLEditorKit, which is provided by Java 7's standard library.

This browser accepts HTML Version 3.2 or older; therefore, any HTML to be added the user must be compatible with this version. Any HTML that is constructed to use features newer than V3.2, may not be rendered as expected. It is important to know that the `<applet>` tag is not supported, but some support is provided for the `<object>` tag.

For more information on HTMLEditorKit, visit the Oracle website: http://docs.oracle.com/javase/7/docs/api/javax/swing/text/html/HTMLEditorKit.html

Help Documentation Methods

This topic provides information on the two methods that can be used to create Help content.

Description

Two methods exist for creating Help content:

- Raw text in the "---help---" section of the configuration entry in the hconfig file, or
- HTML Help, identified by an entry in the MPLAB Harmony Help HTML alias header file

To utilize the first method of providing help content for a library, simply include the appropriate help content for each configuration item in text form in the associated help section for that item in the library's hconfig file.

To utilize the second method, define the appropriate HTML help content in an HTML file. Copy that file into the `<install-dir>/doc/html` folder. Then, append the appropriate Help link (following the conventions described in the following sections) to the end of the HTML alias header file. The order of the entries in the alias header file is not important as it is read, sorted, and searched in it's entirety, by the MHC. However, every alias identifier in the file must be unique, as described in the following section.

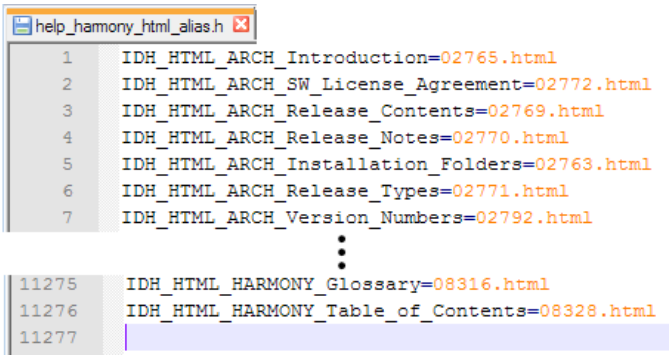
HTML Alias Header File

This topic provides information on the structure and conventions to be followed when adding HTML references to the MHC HTML alias header file.

Description

The HTML alias header file, `help_harmony_html_alias.h`, is located in the following folder within the MPLAB Harmony installation: `<install-dir>/doc/html/`

An example of this file is shown in the following figure:



To add your own HTML file references to this list, use the following conventions:

`IDH_HTML_<NAME>_<ID>_<TopicTitle>=<NAME><file>.html`

Where:

- **`<NAME>`** is an abbreviated company name. For example, IBC, which stands for a company named: Itty Bitty Computer
- **`<ID>`** is the tool identifier. For example, GRC for Graphics Resource Converter.
- **`<TopicTitle>`** is a unique topic identifier. For example, Release_Notes.
- **`<file>`** is the file name (after the company name prefix) of the HTML file for the particular topic

For example, to add a new section named New Tool with a title of New Tool Help to the existing HTML Help, the recommended entry in the alias header file would be:

`IDH_HTML_IBC_TOOL_New_Tool=IBC_new_tool_help.html`

Notes:

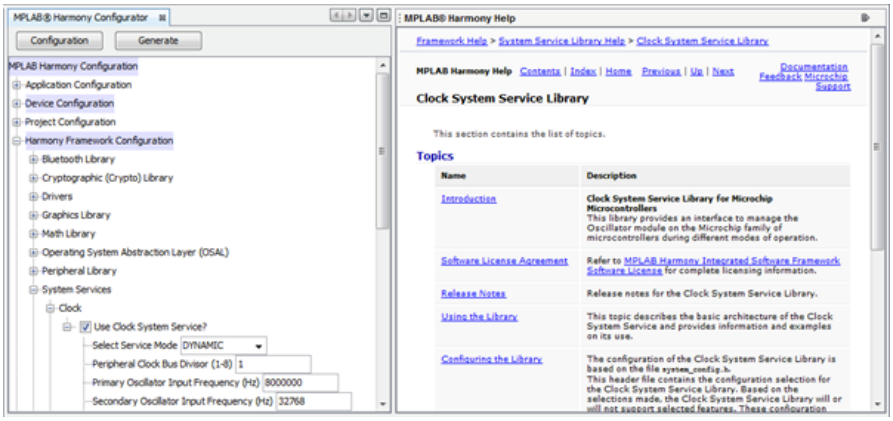
1. The content of your HTML files must be compatible with HTML Version 3.2 or older. The HTML browser used by MHC cannot process HTML tags that are newer than v3.2.
2. You must ensure that any entries added to the existing alias header file are unique from all other entries.
3. When choosing the TopicTitle, use underscores in place of spaces, hyphens, etc.
4. To avoid conflicts with the HTML file numbering used by the MPLAB Harmony Help, it is suggested to use names such as, `IBC_Release_Notes.html`.
5. Add new entries to the end of the file.
6. The following HTML file names are already used by the MPLAB Harmony Help and cannot be reused:
   - `contents.html`
   - `frames.html`
   - `ftxtsearch.html`
   - `header.html`
   - `idx.html`
   - `index.html`

hconfig Files

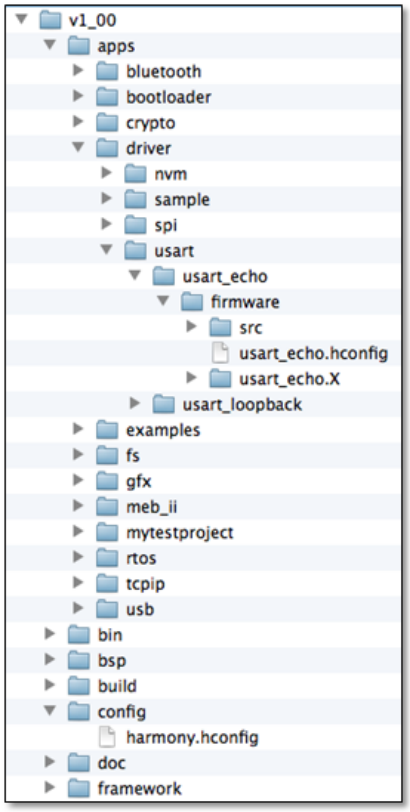
This topic provides information and the location of hconfig files.

Description

The hconfig file tree represents a hierarchy of configuration options presented, with associated Help documentation, by the MHC so that the user can select and configure the desired build options.



Within the MPLAB Harmony installation, hconfig files are kept in the `config` folder at each level in the installation hierarchy that requires them, with one exception. The root of the hconfig file tree is an application-specific file (`<application-name>.hconfig`) that is generated in the project's `firmware` folder. It is not a predefined file. This generated hconfig root file enables the creation of application-specific options if desired (see **Note**). The root file defines the "MPLAB Harmony configuration" main menu item and then includes (AKA "sources") the installation's top-level hconfig file (`<install-dir>/config/harmony.hconfig`) for the installed libraries and templates (as illustrated in the following figure). The top-level hconfig file then includes (sources) the next level of hconfig files in the hierarchy, each of which includes the next level, and so on, down to the individual library hconfig files, which form the "leaves" of the hconfig tree.



Important! The MHC does not currently provide a graphical method of creating application-specific configuration options. It is therefore necessary to manually edit the application-specific hconfig file to create application-specific configuration options that will appear in the MHC tree.

Kconfig Language Specification

This topic provides information on obtaining the Kconfig Language Specification.

Description

The MHC hconfig grammar is based on the Linux Kconfig language specification, with a number of MHC-specific extensions. Please reference the following link for documentation of the core Kconfig language specification. The hconfig extensions are documented in the next section.

https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt

hconfig Language Extensions (Kconfig+)

This sections provides information on the extensions that have been added to the Kconfig grammar to form the hconfig grammar.

"enum"

This topic describes the "enum" extension.

Description

Syntax: `"enum" <enum set name> [exclusive] <string> [|| <string>]...`

The enum entry specifies a named set of possible input values for string symbols. The enum set name can be used within a string range attribute. The optional 'exclusive' attribute indicates that each config symbol that references the enumeration must use a unique enum string value. Symbols that have been used are grayed out in the combo box drop down list, although they can still be selected. Multiple uses of an exclusive enum value will be flagged as an error.

The keyword "enum" both starts and ends a menu entry.

Example:

```
enum PLIB_MODULE_ID exclusive
"PLIB_ID_0"
|| "PLIB_ID_1"
|| "PLIB_ID_2"
```

"range"

This topic describes the "range" extension.

Description

Syntax: **`"range" <enum set name> ["if" <expr>]`**

The string range attribute specifies the set of possible values for a string symbol. The user can only input one of the enumerated values of the enum set names. Any default value must be included in these enumerated values.

Example:

```
config PLIB_MODULE
string "PLIB Module"
range PLIB_MODULE_ID
default "PLIB_ID_0"
```

"template"

This topic describes the "template" extension.

Description

Syntax: **`"template" <template name> <template file path> to <project logical path> ["if" <expr>]`**

The template entry specifies a file to be processed as a FreeMarker template file and copied to a specific location within the project logical path structure.

Example:

```
template SYSTEM_CONFIG_H
"$HARMONY_VERSION_PATH/utilities/mhc/templates/app/system_config/system_config.h.ftl" to
                        "$PROJECT_HEADER_FILES/system_config/$CONFIGURATION/system_config.h"
```

"file"

This topic describes the "file" extension.

Description

Syntax: **`"file" <file name> <file path> [to <project logical path>] ["if" <expr>]`**

The file entry specifies a file name to be added into the project structure. The path to the file is normally added to the project source search paths. However, if the [`to <project file path>`] is specified, the file is physically copied into the project logical path structure.

Example:

```
file DRV_USART_H "$HARMONY_VERSION_PATH/framework/driver/usart/drv_usart.h" to
                 "$PROJECT_HEADER_FILES/framework/driver/usart/drv_usart.h
```

"library"

This topic describes the "library" extension.

Description

Syntax: `"library" <library name> <library file path> ["if" <expr>]`

The library entry specifies a library to be added to the Project linker directives. The path to the library is added to the Project library search paths.

Example:
```
library DEVICE_PERIPHERALS_A "$HARMONY_VERSION_PATH/bin/framework/peripheral/$DEVICE_peripherals.a"
```

"execute"

This topic describes the "execute" extension.

Description

Syntax: `"execute" <exec name> <plugin name> "if" <expr>`

Whenever the "if expression" changes value from *false* to *true*, MHC immediately executes an asynchronous plug-in. The "if expression" must transition from *true* to *false* to *true* again to force another execution of the plug-in.

Example:
```
execute GDDX_PLUGIN GDDX if USER_EXECUTES_GDDX
```

"persistent"

This topic describes the "persistent" extension.

Description

The persistent attribute indicates that the symbol cannot be modified by the user.

Syntax: `"persistent" ["if" <expr>]`

Example:
```
config PERS
bool "Make persistent"
default y

config SOME_INT
int "Enter an int for $PROJECT_NAME in $DEVICE"
default 0
persistent if PERS
```

hconfig Environment Variables

This topic provides information on the hconfig environment variables.

Description

Within the hconfig language, environment variables may be used to reference more global MPLAB X IDE project information. These environment variables, which begin with a "dollar sign" ($), are by convention uppercase, and function much like C preprocessor variables.

The hconfig environment variables include:

Variable Name	Description
`$HARMONY_VERSION_PATH`	Physical pathname to the MPLAB Harmony directory (i.e., `C:/microchip/harmony/<version>`).
`$PROJECT_NAME`	MPLAB X IDE main project name when the Generate option was selected.
`$PROJECT_FIRMWARE_DIRECTORY`	Physical path to the project's `firmware` directory.
`$PROJECT_BSP_DIRECTORY`	Physical path to the project's `bsp` directory.
`$PROJECT_HEADER_FILES`	Logical path to the project header files.
`$PROJECT_SOURCE_FILES`	Logical path to the project source files.
`$CONFIGURATION`	MPLAB X IDE project configuration name.
`$DEVICE`	MPLAB X IDE project device name.

$OS_NAME	Name of the operating system on the computer running MPLAB X IDE.

hconfig Configuration Variables

This topic provides information on the hconfig configuration variables.

Description

The following table lists available hconfig configuration variables:

Variable Name	Description
DEVICE	Supplies the device variant ID in string format.

The following is an example of how these variables are used:

CONFIG_DEVICE

This hconfig symbol can be used to provide the device ID based on the device selected in MPLAB X IDE. This feature is useful if hconfig/FTL logic that is unique to a device variant needs to be added.

The following example shows the FTL code to perform a check for a specific device:

```
<#if CONFIG_DEVICE == “PIC32MZ2028ECM144”>
  … perform device-specific code …
<#/if>
```

Complete hconfig Grammar Definition

This topic provides a complete listing of the hconfig grammar definition.

Description

```
Model:
 ( Statements += Statement)*;

Statement:
   CommonStatement
 | MainmenuStmt
 | MenuStmt
 | ChoiceStmt
;

CommonStatement:
      IfStmt
    | CommentStmt
    | ConfigStmt
    | MenuconfigStmt
    | SourceStmt
    | EnumStmt
    | TemplateStmt
    | FileStmt
    | LibraryStmt
    | ExecuteStmt
    | CompilerStmt
 | AssemblerStmt
;

TemplateStmt:
    'template' name= ID templateFilePath=STRING 'to' templateLogicalPath=STRING ( 'if' (Expr = Expr) )?
;

FileStmt:
    'file' name=ID filePath=STRING ('to' fileLogicalPath=STRING)? ( 'if' (Expr = Expr) )?
;

LibraryStmt:
    'library' name=ID libraryPath=STRING ( 'if' (Expr = Expr) )?
;
```

```
ExecuteStmt:
    'execute' name=ID
    (OptionList += Option*)
    // Only valid execute options are Prompt | Dependency | Default | HelpText=KCONFIG_HELP
;

CompilerStmt:
 'compiler' name=ID  which=('C' | 'CPP')?  type=('define' | 'undefine' | 'includepath') str=STRING ( 'if'
(Expr = Expr ) )?
;

AssemblerStmt:
   'assembler' name=ID  type=('define' | 'undefine' | 'includepath') str=STRING ( 'if' (Expr = Expr ) )?
;
IfStmt:
    'ifblock' ifexpr=Expr
    (statements += Statement*)
    'endif'
;

MainmenuStmt:
    'mainmenu' value=STRING
;

MenuStmt:
    'menu' value=STRING
    (VisibilityList += Visible*)
    (DependsList    += Dependency*)
    (Helptext = KCONFIG_HELP)?
    (MenuBlockList  += Statement*)
    'endmenu'
;

Visible:
    Visible = 'visible' ( 'if' (visible_expr = Expr) )?
;

Dependency:
    'depends on' depexpr = Expr
;

MenuconfigStmt:
    'menuconfig' name= ID
    (OptionList += Option*)
;

CommentStmt:
    'comment' value=STRING
    (DependsList += Dependency*)
    (Helptext = KCONFIG_HELP)?
;

EnumStmt:
    'enum' name=ID (exclusive='exclusive')?
    (Firststring = STRING)
    (Orstrings += Orstring)*
    (Helptext = KCONFIG_HELP)?
;

Orstring:
    '||' value=STRING
;

ChoiceStmt:
    ChoiceStmt = 'choice' (name = ID)?
    (OptionList += ChoiceOption*)
    (Helptext = KCONFIG_HELP)?
    (statements += ConfigStmt*)
    'endchoice'
```

```
;

ChoiceOption:
    Optional | Prompt | Dependency | Default
;

Optional:
    Optional='optional'
;

Option:
    Type | Prompt | Range | Dependency | Select | Default | Persistent | MiscOption | HelpText=KCONFIG_HELP
;

SourceStmt:
    'source' path=STRING (numInstances=SIGNED_INT 'instances')?
;

ConfigStmt:
    'config' name= ID
    (OptionList += Option*)
;

Type:
    type=('bool'|'tristate'|'int'|'hex'|'string') tprompt=STRING? ('if' ifexpr=Expr)? |
    type=('def_bool'|'def_tristate') defexpr=Expr ('if' ifexpr=Expr)?
;

Select:
    'select' name=ID ('if' ifexpr = Expr)?
;

Set:
    'set' name=ID 'to' value=Expr ('if' ifexpr = Expr)?
;

Default:
    'default' (value=Expr) ('if' ifexpr = Expr)?
;

Persistent:
    persistent='persistent' ('if' ifexpr = Expr)?
;

Prompt:
    'prompt' value=STRING ('if' ifexpr = Expr)?
;

Range:
    'range' rangeexpr=RangeExpr ('if' ifexpr = Expr)?
;

MiscOption:
    'option' (MiscOption='modules' | MiscOption='allnoconfig_y' | MiscOption='env' '=' string=STRING |
MiscOption='defconfig_list')
;

RangeExpr returns KconfigExpr:
     RangeLiteral ({RangeExpr.left=current} right=RangeLiteral)?
;

RangeLiteral:
    (conf = ID | signed_int=SIGNED_INT | hex=HEX_TERMINAL)
;

Expr returns KconfigExpr:
    OrLiteral ({Expr.left=current} '&&' right=OrLiteral)*
;
```

```
OrLiteral returns KconfigExpr:
    EqLiteral ({OrLiteral.left=current} '||' right=EqLiteral)*
;

EqLiteral returns KconfigExpr:
    NeqLiteral ({EqLiteral.left=current} '=' right=NeqLiteral)?
;

NeqLiteral returns KconfigExpr:
    PrimaryLiteral ({NeqLiteral.left=current} '!=' right=PrimaryLiteral)?
;

PrimaryLiteral returns KconfigExpr:
    ConfigLiteral | NotLiteral | NotExpr | ParenExpr
;

NotExpr:
    '!' '(' NotExpr=Expr ')'
;

ParenExpr:
    '(' ParenExpr=Expr ')'
;

NotLiteral:
    '!' (NotLiteral = ID)
;

ConfigLiteral:
    conf = ID | signed_int = SIGNED_INT | hex = HEX_TERMINAL | string = STRING
;


terminal ID:
    ('1'..'9')('0'..'9')('0'..'9')('0'..'9')('0'..'9')
        (('A'..'Z')|('a'..'z')|'_')
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
    |

     ('1'..'9')('0'..'9')('0'..'9')('0'..'9')
         (('A'..'Z')|('a'..'z')|'_')
         (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
     |

    ('0'..'9')('0'..'9')('0'..'9')
        (('A'..'Z')|('a'..'z')|'_')
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
     |

     ('1'..'9')('0'..'9')
         (('A'..'Z')|('a'..'z')|'_')
         (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
     |

    ('1'..'9')
        (('A'..'Z')|('a'..'z')|'_')
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
    |

    (('A'..'Z')|('a'..'z'))
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
    ;

terminal HEX_TERMINAL: '0x'('0'..'9'|'a'..'f'|'A'..'F')*;
terminal KCONFIG_HELP: ('---help---' | 'help') ('\r'? '\n') -> '---endhelp---' ('\r'? '\n');
terminal SL_COMMENT: '#' !('\n'|'\r')* ('\r'? '\n')?;
SIGNED_INT: ('-')? INT;
```

Please refer to Kconfig Language Specification and hconfig Language Extensions (Kconfig+) for semantic descriptions of the hconfig grammatical elements. For usage information, refer to Developing a New hconfig File.

MHC Files

This topic provides an example MHC file.

Description

The MHC stores the user's selections in an MHC file. An MHC file is created for each configuration, named using the configuration name provided by the MPLAB-IDE, and located (by default) in the configuration-specific `system_config` folder within the `src` folder in the default MPLAB Harmony project.

Default MHC file name and location: `<my_project>/firmware/src/system_config/<my_config>/<my_config>.mhc`

The MHC file is analogous to the `.config` file in a Linux system configuration. It is created and maintained by the MHC and should not be edited by the user. It is parsed when the user clicks **Generate** within the MHC configuration window to provide the data set utilized by the FreeMarker engine when processing the MPLAB Harmony template (`.ftl`) files. This file captures all settings created by user selections in the MHC GUI and can be shared or copied to duplicate a complete set of configuration selections.

MHC prepends `CONFIG_` to each config option, and stores the value in the `.mhc` file. The format is:

`CONFIG_<config option>=<value>`

A common mistake when creating FreeMarker templates is to forget the leading `CONFIG_` when using config values to generate code.

The following example shows `.mhc` file entries for the example drivers that were used in Developing a New hconfig File:

```
93 #
94 # from $HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_example.hconfig
95 #
96 CONFIG_USE_HCONFIG_EXAMPLE=y
97 CONFIG_CFG1="cfg1_val_0"
98 CONFIG_CFG2=y
99 CONFIG_CFG3=1
00 CONFIG_CFG4=n
01 CONFIG_CFG6=n
02 CONFIG_CFG7=y
03 CONFIG_CFG8=n
04 CONFIG_CFG9=y
05 CONFIG_CFG10=y
06 CONFIG_DRV_HCONFIG_INSTANCES_NUMBER=3
07 #
08 # from $HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_idx.ftl
09 #
10 CONFIG_DRV_HCONFIG_INST_IDX0=y
11 CONFIG_DRV_HCONFIG_CFG12_IDX0=n
12 CONFIG_DRV_HCONFIG_CFG13_IDX0=42
13 CONFIG_DRV_HCONFIG_INST_IDX1=y
14 CONFIG_DRV_HCONFIG_CFG12_IDX1=y
15 CONFIG_DRV_HCONFIG_CFG13_IDX1=43
16 CONFIG_DRV_HCONFIG_INST_IDX2=y
17 CONFIG_DRV_HCONFIG_CFG12_IDX2=n
18 CONFIG_DRV_HCONFIG_CFG13_IDX2=42
```

Note: The `.mhc` file does not contain config-option definitions for modules that are not selected for use. However, keep in mind that a module may be selected for use by default or as a result of the selection of another module that requires it.

MHC Configuration File

This topic describes the purpose of the `configuration.xml` file.

Description

The file, `configuration.xml`, is used by the MHC to store configuration-specific information. The `configuration.xml` file is created by the MHC for all managed configurations. This file resides in the configuration's `system_config` folder.

The information that this file currently contains includes:

- The configuration's MPLAB Harmony path
- The configuration user preferences
- A list of automatically added files (untracked)
- A list of automatically added templates (tracked)
- A list of automatically added libraries

The tracked attribute means that the generated file is being tracked using checksums.

If this file is not present, the MHC will prompt the user for a MPLAB Harmony path. The file will then be recreated. Upon configuration regeneration, the MHC will compare existing files to the list of generated files. If a name match occurs, the user will be prompted to merge the two files.

Important! This file is automatically generated by the MHC and should not be manually modified.

BSP XML Specification

This topic describes the format of the `bsp.xml` file, which is required for MHC Board Support Package (BSP) development.

Description

The `bsp.xml` file contains pin information pertinent to an individual Board Support Package or BSP. MHC uses this file to add the appropriate options to the Pin Manager table during configuration. When a BSP is properly organized and presented, MHC will find the appropriate file and dynamically load it when the BSP is selected in the HConfig tree.

This file must reside in the `xml` sub-folder within the desired BSP folder. The XML file must be named `bsp.xml`. An example path for the BSP that supports the PIC32 Bluetooth Audio Development Kit would be: `<install-dir>/bsp/bt_audio_dk/xml/bsp.xml`.

File Example

The following example shows what this `bsp.xml` file might contain:

```
<?xml version="1.0"?>
<bsp name="bt_audio_dk">
    <function name="SWITCH_1" pin="RA0" mode="digital" pullup="true"/>
    <function name="SWITCH_2" pin="RA1" mode="digital" pullup="true"/>
    <function name="SWITCH_3" pin="RA10" mode="digital" drain="true" pullup="true"/>
</bsp>
```

The root node is named 'bsp' and contains a name attribute unique to this package. The root node contains any number of child nodes defining functions that this BSP will add to the Pin Manager table.

The function node must have these required attributes:

- name – a custom name assigned to this function
- pin – the pin name to which this function is attached

The function node may also have these attributes:

- direction – 'in' or 'out', default = 'in'
- latch – 'high' or 'low', default = 'low'
- drain – 'true' or 'false', default = 'false'
- mode – 'digital' or 'analog', default = 'analog
- cn – 'true' or 'false', default = 'false'
- pullup – 'true' or 'false', default = 'false'
- pulldown – 'true' or 'false', default = 'false'

When a BSP is added in HConfig, these defined values will be pushed to the corresponding pin. If it is removed, the pin will return to its default state. The Pin Manager does not prevent the user from changing these values in the Pin Manager after they have been read from the XML file.

Note: To ensure that alterations to `bsp.xml` files are applied, developers must manually clear and reselect the corresponding BSP entry in HConfig. This will notify MHC to reapply the xml values.

Adding New BSPs

This section provides information adding a new BSP.

Updating the BSP hconfig File

This topic provides information on configuring the hconfig file for the purpose of adding a new BSP.

Description

Adding a new Board Support Package (BSP) is a three-step process, which includes:

- Updating `<install-dir>/bsp/config/bsp.hconfig` with the new BSP
- Creating a new `bsp` folder with the necessary BSP files
- Updating `<install-dir>/bsp/config/bsp.config` with the path to the new `bsp.hconfig` file within your new `bsp` directory

Step One: Within the `choice` statement, create and name the `bool` config for the new BSP and specify (*the first three items are required, the fourth is optional*):

- Upon which device family this BSP depends
- The dependency on USE_BSP

- The BSP_TRIGGER selection
- Optionally, which MPLAB Harmony components this BSP should select (i.e., enable)

For example, if a new BSP uses a PIC32MZ EF device, which needs to enable the Graphics Library, the hconfig code may appear like the following:

```
config BSP_MYBOARD
   depends on USE_BSP
   depends on DS60001320    # Microchip document number for devices that can use this BSP
   select BSP_TRIGGER
   select USE_GFX_STACK
   bool "BSP for my board"  # Ensure you are using the correct quotation marks to prevent errors
```

Step Two: Specify which files should be added to the MPLAB X IDE project when the new BSP is selected, as well as the include path. Outside of the `choice` statement, define a new `ifblock` statement, as follows:

The easiest way to create the files required is to copy an existing `bsp` folder structure from within `<install-dir>/bsp` and edit the files accordingly. There are four files that you will need to edit:

- Files 1 and 2 - `bsp_config.h` and `bsp_sys_init.c` are the files that will be included in your project. These files include the macros and defines for use within your code.
- File 3 - The XML file described below that has your pin descriptions (copy an existing XML for formatting)
- File 4- The `bsp.hconfig` file within your `bsp` directory that contains the following information:
  - Path to XML file containing the pin description for the new BSP (see BSP XML Specification for more information)
  - Path to `bsp_config.h` file
  - Path to `bsp_sys_init.c` file
  - Compiler include path

For example, if the new BSP uses a PIC32MZ EF device that needs to enable the Graphics Library, the hconfig code may appear similar to the following:

```
ifblock BSP_MYBOARD
  file BSP_my_board_xml "$HARMONY_VERSION_PATH/bsp/my_board/xml/bsp.xml" to "$BSP_CONFIGURATION_XML"
  file BSP_ my_board _H "$HARMONY_VERSION_PATH/bsp/my_board/bsp_config.h" to
"$PROJECT_HEADER_FILES/bsp/my_board/bsp_config.h"
  file BSP_my_board_C "$HARMONY_VERSION_PATH/bsp/my_board/bsp_sys_init.c" to
"$PROJECT_SOURCE_FILES/bsp/my_board/bsp_sys_init.c"
  compiler BSP_COMPILER_INCLUDE_my_board includepath "$HARMONY_VERSION_PATH/bsp/my_board "
endif
```

Step Three: Add a pointer to your new BSP in the `<install-dir>/bsp/config/bsp.hconfig` file. Your new line will be at the end of the file and should appear similar to the **bold** line in the following example:

```
...
source "$HARMONY_VERSION_PATH/bsp/pic32mz_ef_sk+meb2+wvga/config/bsp.hconfig"
source "$HARMONY_VERSION_PATH/bsp/pic32mz_ef_sk+s1d_pictail+vga/config/bsp.hconfig"
source "$HARMONY_VERSION_PATH/bsp/pic32mz_ef_sk+s1d_pictail+wqvga/config/bsp.hconfig"

source "$HARMONY_VERSION_PATH/bsp/my_board/config/bsp.hconfig"
endmenu
```

MPLAB Harmony Configurator Plug-ins

Describes the clock screen system plug-in interface.

Description

MPLAB Harmony Configurator provides a plug-in interface into the clock screen system.

System Requirements

The system requirements for a MPLAB Harmony Configurator clock screen plug-in are:

- NetBeans v8.0 or later
- Java 7
- MPLAB X IDE v3.06 or later
- MPLAB Harmony Configurator v1.06 or later
- A Java JAR file containing a class that inherits from the abstract class “com.microchip.mplab.modules.mhc.clock.ClockModel”.

NetBeans Project Setup

Java Dependencies

Your project will have a dependency on the library `com.microchip.mplab.modules.mhc.jar`. Once MPLAB X IDE and the MPLAB Harmony Configurator have been installed, this file can be found in the following Windows location:
`C:\Users\($YOUR_USER_NAME)\AppData\Roaming\mplab_ide\dev\($MPLABX_VERSION)\modules`.

Please reference the example clock screen plug-in project for more detailed programming interface information.

The project can be found in the MPLAB Harmony framework within the `<install-dir>\utilities\mhc\plugins\clock\plugin_example` folder.

Plug-in Installation:

There are two steps required to install your plug-in into MHC.

1. *Plug-in file*. NetBeans will produce a JAR file of your plug-in. This file must be copied to the MPLAB Harmony framework folder: `<install-dir>\utilities\mhc\plugins\clock`.
2. *HConfig* - MPLAB Harmony Configurator relies on the HConfig tree to tell it which clock plug-in file to load. Often this is processor-specific. To get your plug-in loaded, you must edit the MPLAB Harmony framework file: `<install-dir>\framework\config\framework.hconfig`.
   - The string symbol `SYS_CLK_MANAGER_PLUGIN_SELECT` must be defined. This symbol value must have the following format: `($JAR_FILE_NAME):($CLOCK_MODEL_CLASS)`, where:
     - `($JAR_FILE_NAME)` - the name of your plug-in JAR file without the `.jar` file extension
     - `($CLOCK_MODEL_CLASS)` - the name of the class in your plug-in that inherits from the ClockModel base class
   - For example, to load the MX1 clock screen the symbol would be set to: `mx1:MX1ClockModel`

Once these two items are complete, MHC will attempt to load your clock plug-in at start-up. If loading fails, an exception and stack trace will be printed.

Debugging Plug-ins

MPLAB X IDE Configuration

MPLAB X IDE can be configured to allow NetBeans to debug MPLAB Harmony Configurator plug-ins, as follows:

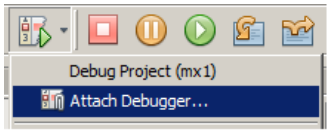
1. Open the following file in a text editor: `($MPLABX_ INSTALLATION_PATH)/($MPLABX_VERSION)/etc/mplab_ide.conf`.
2. Locate the configuration entry `default_options`. Add the following text to the line (without a line break):

```
-J-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5858
```

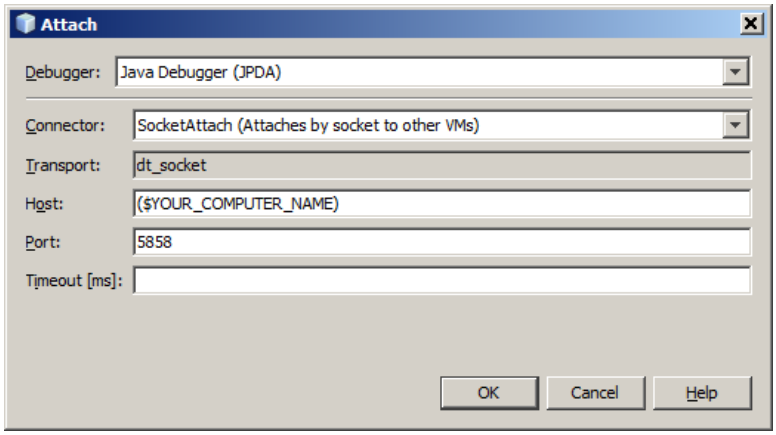
This instructs MPLAB X IDE to allow debugging over the socket 5858.

NetBeans Configuration

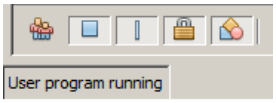
1. To attach to MPLAB X IDE, right-click the Debug Project drop-down menu and select **Attach Debugger**.



2. Configure the Attach dialog, as shown in the following figure.



3. If MPLAB X IDE is running and you configured everything properly, the message “User program running” should appear in the lower-left corner of NetBeans.



You can now set breakpoints in your plug-in code and debug as normal. If you receive the message *Connection Refused*, this indicates that something has been misconfigured.

Pin Manager Development

Provides details on pin manager development.

Description

The MPLAB Harmony Configurator Pin Manager system is a data driven state machine that provides the capability for users to configure the I/O

pins for many different components. It also provides a data-driven mechanism for drawing basic representations of these components.

The following table provides common terms and their descriptions.

Term	Description
Pin Manager	A system for configuring component I/O pins.
Pin Diagram	A visual representation of a component.
Pin Table	A matrix-based system for assigning functions to pins.
Pin	A single I/O interface on a component.
Package	A physical pin layout for a component. Components may come in several packages.
Function	A processing capability that a component supports (e.g., UTX1).
Module	Designates a set of related functions (e.g., UART1).
Component	A discreet part number (e.g., PIC32MX110F016B).
Family	Designates a superset of components (e.g., PIC32MZEF).

File Parsing

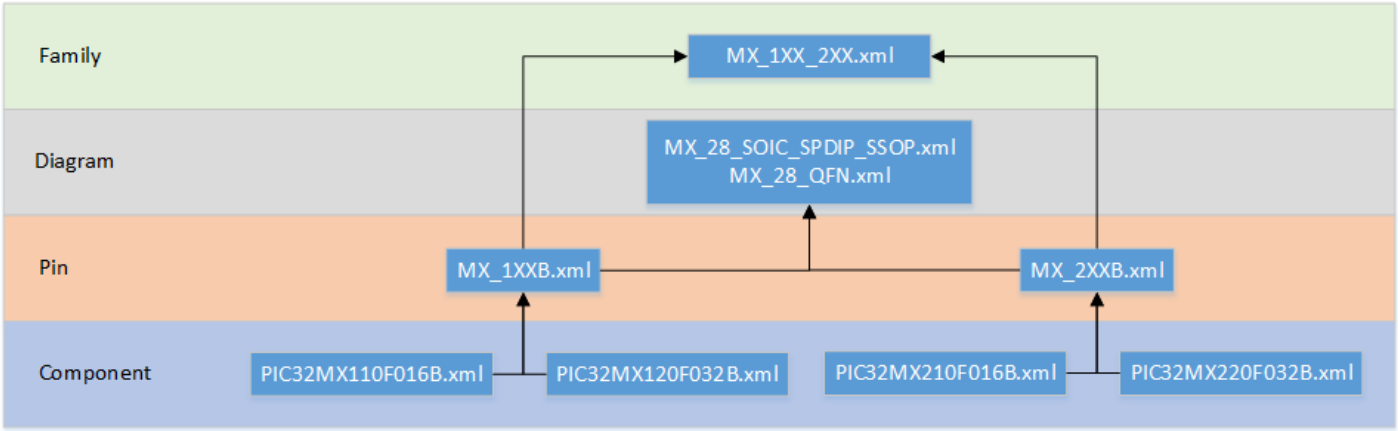
The pin manager is responsible for parsing through a set of XML files for the purpose of building a component pin state. These data files are located in the MPLAB Harmony framework within `<install-dir>/utilities/mhc/pin_xml`.

These data files come in four types, Component, Pin, Diagram, and Family:

- **Component** – A unique file for every component supported by MPLAB Harmony Configurator. This data file links the component to a pin file.
- **Pin** – A file that describes the physical characteristics of a component, which includes:
  - Available Packages
  - Pin-to-Package Association – This is needed because a function may not map to the same pin numbers for every package
  - Package-to-Diagram association
  - Supported pin functions – The function groups do not change between packages; however, their associated pin may change
- **Diagram** – A file that describes how to render an image of the selected component package.
- **Family** – Provides several different functions:
  - PPS information (if available, which is taken directly from the product data sheet)
  - Module information:
    - Instructs the pin manager as to how to group available functions in the pin table
    - Provides the capability to specify display constraints, which is what allows the pin table to show UART1 when the UART driver is enabled in the option tree
    - Allows the capability to specify module and function characteristics.

XML File Hierarchy

The following diagram provides a visual illustration of the XML file hierarchy.



Detailed File Descriptions

Component

The component file only has one entry that maps the selected component to its pin map file.

```
<component device="PIC32MX110F016B" pins="MX_1XXB" />
```

Pin

The `pinfile` root node of the pin file maps the pin file to the family file:

```
<pinfile family="MX_1XX_2XX">
```

The `pinfile` node has two main child nodes: `packages` and `pins`

One or more `package` nodes will be listed inside the `packages` node.

```
<packages>
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="1" name="SOIC" />
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="2" name="SPDIP" />
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="3" name="SSOP" />
<package diagram="MX_28_QFN" id="4" name="QFN" />
</packages>
```

`Package Node`

The package node description is as follows:

- `diagram` – designates the diagram for a package
- `id` – a unique numerical identifier. This governs the order in which the package appears in the pin table package selector.
- `name` – the name of this package that will be shown in the pin table package selector

One or more `pin` nodes will be listed inside the `pins` node.

```
<pin name="RB5">
<modifiers>
<modifier value="5V" />
</modifiers>
<number package="1" pin="14" />
<number package="2" pin="14" />
<number package="3" pin="14" />
<number package="4" pin="11" />
<function name="PGED3" />
<function name="RPB5" />
<function name="PMD7" />
</pin>
```

Pin Node

The `pin` node description is as follows:

- `modifiers` – The `modifiers` node can have a list of modifier nodes attached to it. **Note:** Currently, only one modifier “5V” is specified. However, this value is no longer used by the pin manager and will be removed in a future version.
- `number` – provides a map between a pin name, a package, and a pin number within that package
- `function` – provides a list of functions supported by this pin

Diagram

A diagram file instructs the pin diagram rendering engine how to draw the particular package for a selected component.

```
<diagram min_x="380" min_y="380" >
<shape type="rect" width="160" height="160" stroke="2"/>
<shape type="string" line="000000" val="$DEVICE_NAME" orientation="right" size="11"/>
<shape type="circle" x="-75" y="-75" radius="5" stroke="1" fill="000000"/>
<layout type="row">
<row pins="1-7" x="-80" margin="5" direction="down"
pin_width="7" pin_height="10" pin_name_location="left" pin_name_size="10" pin_name_margin="10"
pin_number_location="right" pin_number_size="10" pin_number_margin="6" pin_number_orientation="up" />
<row pins="8-14" y="72" margin="5" direction="right"
pin_width="10" pin_height="7" pin_name_location="down" pin_name_size="10" pin_name_margin="10"
pin_number_location="up" pin_number_size="10" pin_number_margin="5" pin_number_orientation="left" />
<row pins="15-21" x="72" margin="5" direction="up"
pin_width="7" pin_height="10" pin_name_location="right" pin_name_size="10" pin_name_margin="10"
pin_number_location="left" pin_number_size="10" pin_number_margin="6" pin_number_orientation="up" />
<row pins="22-28" y="-80" margin="5" direction="left"
pin_width="10" pin_height="7" pin_name_location="up" pin_name_size="10" pin_name_margin="10"
pin_number_location="down" pin_number_size="10" pin_number_margin="5" pin_number_orientation="left" />
</layout>
</diagram>
```

The `root` node is `diagram` and has two attributes: `min_x` and `min_y`. These values describe the overall area of the diagram and are useful for controlling the blank space around the diagram.

Shape Nodes

Shape nodes (`shape`) instruct the rendering engine to draw basic shapes. The shape attributes are dependent on the required `type` attribute. The available `shape` types and their sub-attributes, are as follows:

line – A line:

- x – (attribute) the x1 position of the line
- y – (attribute) the y1 position of the line
- x2 – (attribute) the x2 position of the line
- y2 – (attribute) the y2 position of the line
- stroke – (attribute) the width of the line
- line – (attribute) the color of the line represented as a hex value RRGGBB

circle – A circle:

- x – (attribute) the x position of the circle's radius
- y – (attribute) the y position of the circle's radius
- radius – (attribute) the radius of the circle in pixels
- stroke – (attribute) the width of the circle line
- line – (attribute) the color of the circle represented as a hex value RRGGBB
- fill – (attribute) the color used to fill in the shape represented as a hex value RRGGBB

rect – A rectangle centered inside the diagram screen:

- width – (attribute) the width of the rectangle in pixels
- height – (attribute) the height of the rectangle in pixels
- rounded – (attribute) a Boolean value to indicate if the rectangle has round corners. Default is "false". Set to "true" to enable.
- arc – (attribute) indicates the radius of the rounded corners. Ignored if rounded is not "true".
- stroke – (attribute) the width of the rectangle lines
- line – (attribute) the color of the rectangle border represented as a hex value RRGGBB
- fill – (attribute) the color used to fill in the rectangle represented as a hex value RRGGBB

complex_rect – A complex rectangle, centered inside the diagram screen, that can have unique corner descriptions:

- width – (attribute) the width of the rectangle in pixels
- height – (attribute) the height of the rectangle in pixels
- corners – (node) a group node indicating the presence of "corner" attributes
  - corner – (node) a node describing a complex rect corner
    - loc – (attribute) the corner being described. Must be "topleft", "topright", "bottomleft", or "bottomright"
    - type – (attribute) the type of complex corner
      - notch – (value) a notched corner
      - round – (value) a rounded corner
    - length – (attribute) the length of the notch in pixels. Used only if type equals "notch"
    - arc – (attribute) the radius of the rounded corner in pixels. Used only if type equals "round"
- stroke – (attribute) the width of the rectangle lines
- line – (attribute) the color of the rectangle border represented as a hex value RRGGBB
- fill – (attribute) the color used to fill in the rectangle represented as a hex value RRGGBB

string – A text string:

- x – (attribute) the x position of the string
- y – (attribute) the y position of the string
- val – (attribute) the value of the string
  - The value $DEVICE_NAME is a special keyword that will print the selected component name
- orientation – (attribute) controls the direction that the string is printed
  - up – (value) print the string rotated counter-clockwise 90 degrees
  - down – (value) print the string rotated clockwise 90 degrees
- size – (attribute) the font size to use
- stroke – (attribute) the width of the text lines
- line – (attribute) the color of the text represented as a hex value RRGGBB

string_array – An array of strings drawn on separate lines top-down or bottom-up:

- vals – (attribute) a list of strings to print delimited by a comma ",". (e.g., A,B,C,D,E)
- orientation - (attribute) controls the direction that the string is printed
  - up – (value) print the string rotated counter-clockwise 90 degrees (default)
  - down – (value) print the string rotated clockwise 90 degrees
- size – (attribute) the font size to use
- margin – (attribute) the amount of space to pad between the strings

Layout Node

The layout node instructs the rendering engine on how to lay out the pins in the diagram. Pins are typically laid out in rows or grids. When rows are used, sub-segments of pins are assigned to individual rows, and rows are placed as necessary in the grid. The pin diagram will automatically

make the cells for each pin interactive when the application is run.

A `layout` node is defined as such with the type attribute being set to either `row` or `grid`:

```
<layout type="row">
</layout>
```

Row Layout

The `row` layout is used to assign pins to individual rows in the diagram. These rows can be placed anywhere but are typically placed on the outline of the shape used to represent the component package. The pin cells in a `row` layout are rectangular.

Row Node

The `row` node provides the capability to specify a pin row. The `row` node has several required attributes:

`pins` – a numerical range specifying what component pins belong to this row. (e.g. “1”, “1-7”, or “A1-A7”)

`margin` – the numerical amount of pixels to pad between each pin cell in this row

`direction` – the direction to draw this row. Valid values are:

- `up` – row is drawn from bottom to top
- `down` – row is drawn from top to bottom
- `left` – row is drawn from right to left
- `right` – row is drawn from left to right

`pin_width` – describes the width of a pin cell

`pin_height` – describes the height of a pin cell

`pin_name_location` – describes which side of the cell in which to draw the pin name text. Valid values are:

- `left` (default)
- `down`
- `right`
- `up`
- `none`

`pin_name_size` – describes the text size of the pin name

`pin_name_margin` – describes the distance to pad the pin name from the pin cell

`pin_number_size` – describes the size of the text used when drawing the pin number

`pin_number_location` – describes the location of the pin number relative to the pin cell

- `left`
- `down`
- `right` (default)
- `up`
- `inside`

`pin_number_orientation` – describes the orientation of the text representing the pin number. Valid values are:

- `left`
- `down`
- `right`
- `up` (default)

Grid Layout

The `grid` layout is used to display a table of pins in a grid-based layout. Pins are laid out in a uniform manner of rows and columns. Pins are displayed as circles instead of rectangles. Pin numbers are contained inside the circle and the pin name is displayed below the circle.

An example of a `grid` layout is as follows:

```
<layout type="grid" pin_margin="40" pin_name_margin="0" pin_rows="18" pin_cols="18" pin_radius="11" />
```

The attributes of a grid layout are as follows:

- **`pin_margin`** – this describes the spacing of the pin circles
- **`pin_name_margin`** – this describes the distance between the pin name and the pin circle
- **`pin_rows`** – the number of pins per row
- **`pin_cols`** – the number of pins per column
- **`pin_radius`** – the radius of the pin circles

Family

Family files provide the method by which the connections between the physical pin descriptions (pin files) and the MPLAB Harmony Pin Manager’s user interface as well as the HConfig symbol tree.

A family file consists of root “family” node. The two main child nodes of a “family” node are “groups” and “modules”.

Groups

A group describes the Peripheral Pin Select (PPS) capabilities of the family. This data is taken directly from the applicable family data sheet's PPS section. The number of XML groups should match the number of PPS groups specified by the data sheet.

Typical PPS descriptions of input and output groups in a product data sheet are shown in the following two figures:

PPS Input Pins

Peripheral Pin	[*pin name*]R Value to RPn Pin Selection
INT4	0000 = RPA0 0001 = RPB3 0010 = RPB4 0011 = RPB15 0100 = RPB7 0101 = RPC7(2) 0110 = RPC0(1) 0111 = RPC5(2) 1000 = Reserved • • • 1111 = Reserved
T2CK	
IC4	
$\overline{SS1}$	
REFCLKI	

PPS Output Pins

RPn Port Pin	RPnR Value to Peripheral Selection
RPA0	0000 = No Connect 0001 = $\overline{U1TX}$ 0010 = $\overline{U2RTS}$ 0011 = SS1 0100 = Reserved 0101 = OC1 0110 = Reserved 0111 = C2OUT 1000 = Reserved • • • 1111 = Reserved
RPB3	
RPB4	
RPB15	
RPB7	
RPC7	
RPC0	
RPC5	

This data is described in XML format as follows:

```
<group id="1">
<pin name="RPA0" value="0"/>
<pin name="RPB3" value="1"/>
<pin name="RPB4" value="2"/>
<pin name="RPB15" value="3"/>
<pin name="RPB7" value="4"/>
<pin name="RPC7" value="5"/>
<pin name="RPC0" value="6"/>
<pin name="RPC5" value="7"/>
<function name="INT4" direction="in"/>
<function name="T2CK" direction="in"/>
<function name="IC4" direction="in"/>
<function name="SS1 (in)" direction="in"/>
<function name="SS1 (out)" direction="out" value="3"/>
<function name="REFCLKI" direction="in"/>
<function name="U1TX" direction="out" value="1"/>
<function name="U2RTS" direction="out" value="2"/>
<function name="OC1" direction="out" value="5"/>
<function name="C2OUT" direction="out" value="7"/>
</group>
```

`id` – (attribute) the number of this group. This corresponds to the group id in the data sheet.

`pin` – (node) describes a pin that is part of this PPS group:

- `value` – (attribute) the register value that is assigned in the input table

`function` – (node) lists pps functions that can be mapped to the listed pins:

- `name` – (attribute) the name of the function
- `direction` – (attribute) specifies if this function is input or output
- `value` – (attribute) register value for this function (output only)

Note: Some pins may have the same name regardless of I/O direction. In this example this case is mitigated by adding a unique prefix (e.g., (in) or (out)). These prefixes may be stripped out during code generation.

Modules

A module allows a mechanism to group functions together under a common name. It also provides the capability to hook into the HConfig symbol tree. This allows the Pin Table to be dynamic and only show modules that have been enabled by the user based on data-defined constraints.

A module contains the superset of all functions for a particular family. It is often the case that a component does not support all of the functionality defined in its respective data sheet. The Pin Manager will discard any functions that are not found in the corresponding pin file. Modules that have no available functions will not be shown in the table.

An example of a module definition is as follows:

```
<module name="UART 1" desc="UART 1\n(USART_ID_1)">
<function name="U1RX">
<constraint type="enable">
<pair key="DRV_USART_USE_RX_PIN_IDX[0-5]" value="USART_ID_1"/>
</constraint>
</function>
</module>
```

XML Specification Descriptions

Detailed descriptions of the module XML specification are as follows:

`module` – (node):

- `name` – (attribute) a unique name for this module
- `desc` – (attribute) a nicely formatted name. This is what will be shown in the pin table. Line breaks are specified by the string “\n”
- `analog` – (attribute) Boolean value indicating that this module and all of its associated functions are not 5 volt tolerant and that they can be configured as analog-capable. Default is “false”.
- `constraint` – (node) indicates that a constraint is placed on this module
  - `type` – (attribute) specifies the type of constraint
  - `enable` – (value) hides this module if the required constraints are not met. Multiple enable constraints may be used in conjunction:
    - `pair` – (node) a key-value pair:
      - `key` – (attribute) the HConfig symbol to test. In the event that multiple symbols in a particular numerical sequence need to be tested a range can be specified. For example, a module can be dependent on the Hconfig symbols `DRV_USART_USE_RX_PIN_IDX` indices 0 through 5. This can be quickly specified as: `DRV_USART_USE_RX_PIN_IDX[0-5]`
      - `value` – (attribute) a string to test the HConfig symbol against. In the case of a Boolean the value to use is either “y” or “n”

`function` – (node):

- `name` – (attribute) the unique name of this function
- `analog` – (attribute) Boolean value indicating that this function is not 5 volt tolerant and that it can be configured as analog-capable. Default is “false”.
- `constraint` – (node) indicates that a constraint is placed on this function
  - `type` – (attribute) specifies the type of constraint:
    - `enable` – (value) hides this function if the required constraints are not met. Multiple enable constraints may be used in conjunction:
      - `pair` – (node) a key-value pair:
        - `key` – (attribute) the HConfig symbol to test. In the event that multiple symbols in a particular numerical sequence need to be tested a range can be specified. For example, a module can be dependent on the Hconfig symbols `DRV_USART_USE_RX_PIN_IDX` indices 0 through 5. This can be quickly specified as: `DRV_USART_USE_RX_PIN_IDX[0-5]`
        - `value` – (attribute) a string to test the HConfig symbol against. In the case of a Boolean, the value to use is either “y” or “n”
    - `debug` – (value) indicates that this function is a debug function. This places special modifiers on the function and it cannot be selected in the pin table.

Note: A special module is defined for use with Board Support Packages. It must have the name “BSP” to be properly identified by the pin manager. This is module is added to when a BSP is selected in the HConfig option tree.

An example of a BSP module is as follows:

```
<module name="BSP" desc="Board Support Package">
<constraint type="enable">
<pair key="USE_BSP" value="y"/>
</constraint>
</module>
```

MPLAB Harmony Display Manager User's Guide

This section provides information on the MPLAB Harmony Display Manager plug-in.

Introduction

This section provides a guided user experience with a step-by-step procedure that can be used to configure the MPLAB Harmony framework and the MPLAB Harmony Display Manager plug-in tool to prototype a new display. The hardware platform used as an example will be the PIC32MZ EC Starter Kit plus the Multimedia Expansion Board II (MEB II).

For hardware support to connect your own display to the MEB II, please contact your local Microchip sales office.

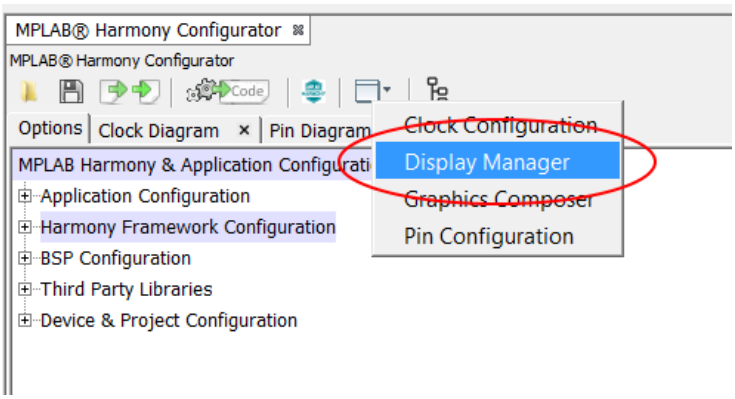
Configuring a New Display

Provides the steps to create a New MPLAB Harmony project for the purpose of rendering a test graphics screen on the display.

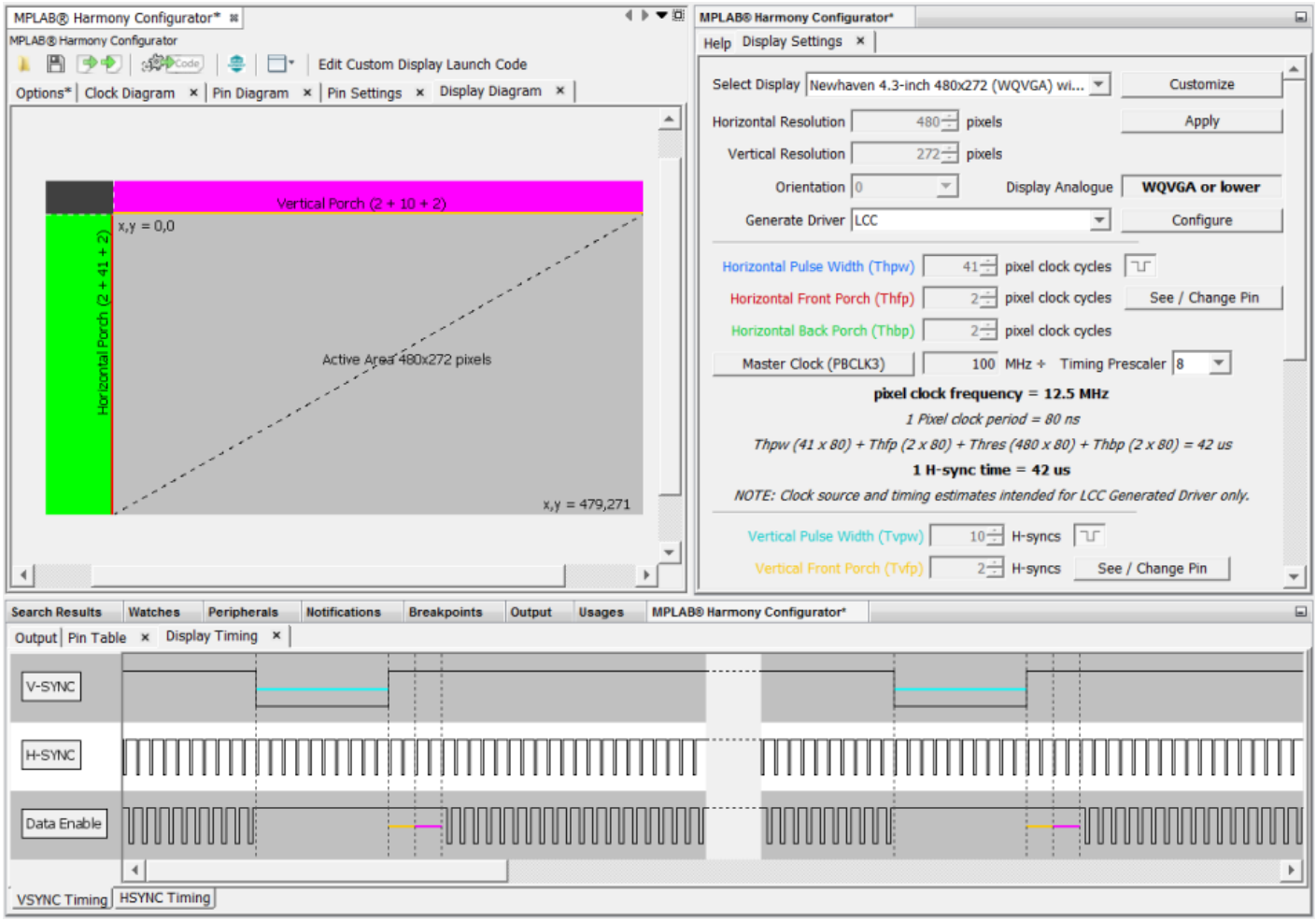
Description

Use the follow process to create a new MPLAB Harmony project to render a test graphics screen on the display:

1. Create a new MPLAB Harmony project using the instructions provided in MPLAB Harmony Configurator User's Guide > Using MHC to Create a New Application > Step 1: Create the New Project and select the PIC32MZ2048EHM144 as the device.
2. The MPLAB Harmony Configurator (MHC) will be launched automatically.
3. Perform the following configuration changes in the MHC Tree:
   - BSP Configuration > select **Use BSP**
   - Selecting BSP to Use:
     - PIC32MZ EC Starter Kit w/Multimedia Expansion Board (MEB) II
     - Graphics Stack > Use Graphics Stack? > set to **Enable**
4. Launch the MPLAB Harmony Display Manager (MHDM) using the plug-in drop-down menu.

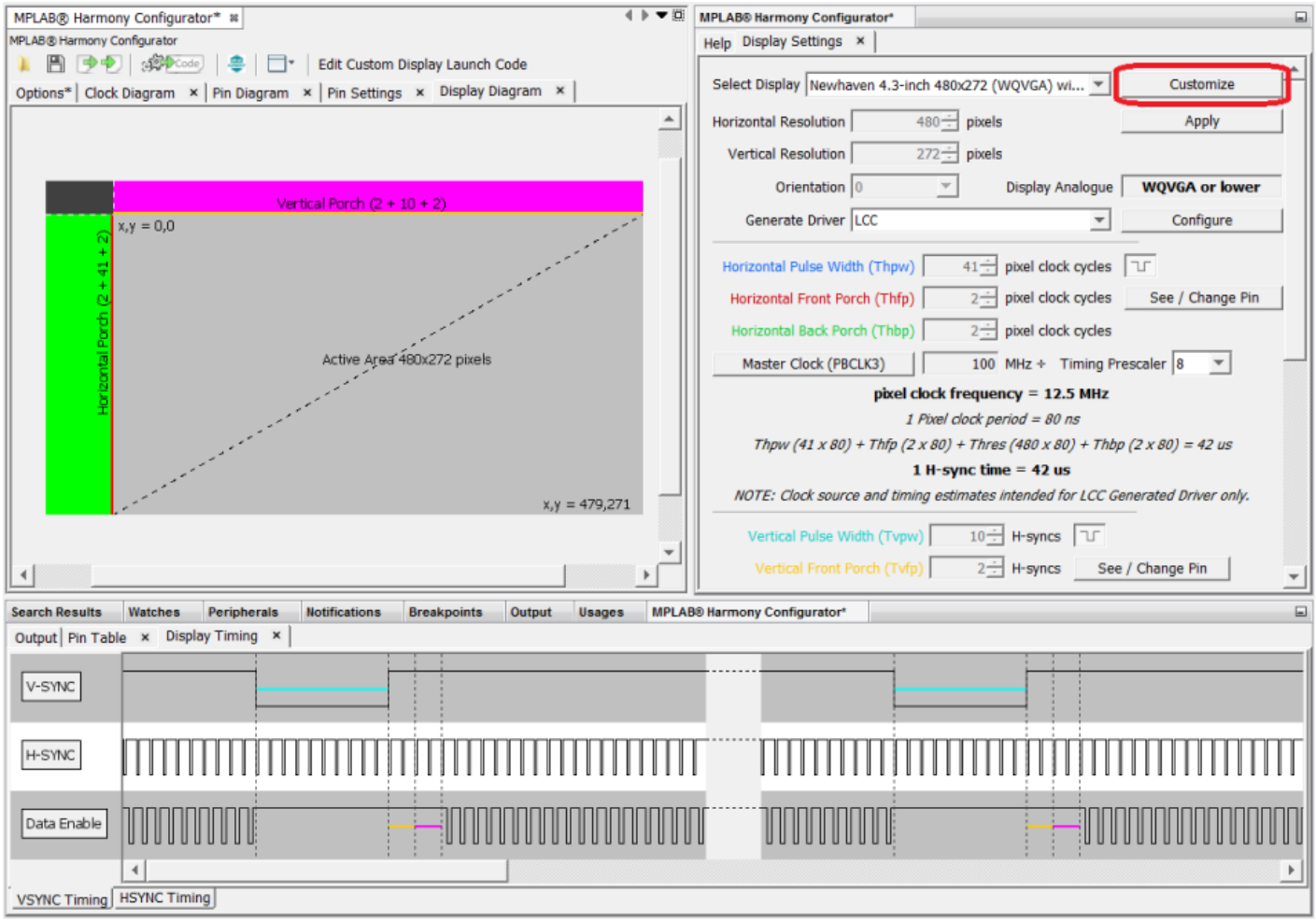


5. The Display Manager will launch and bring its tabs into focus.

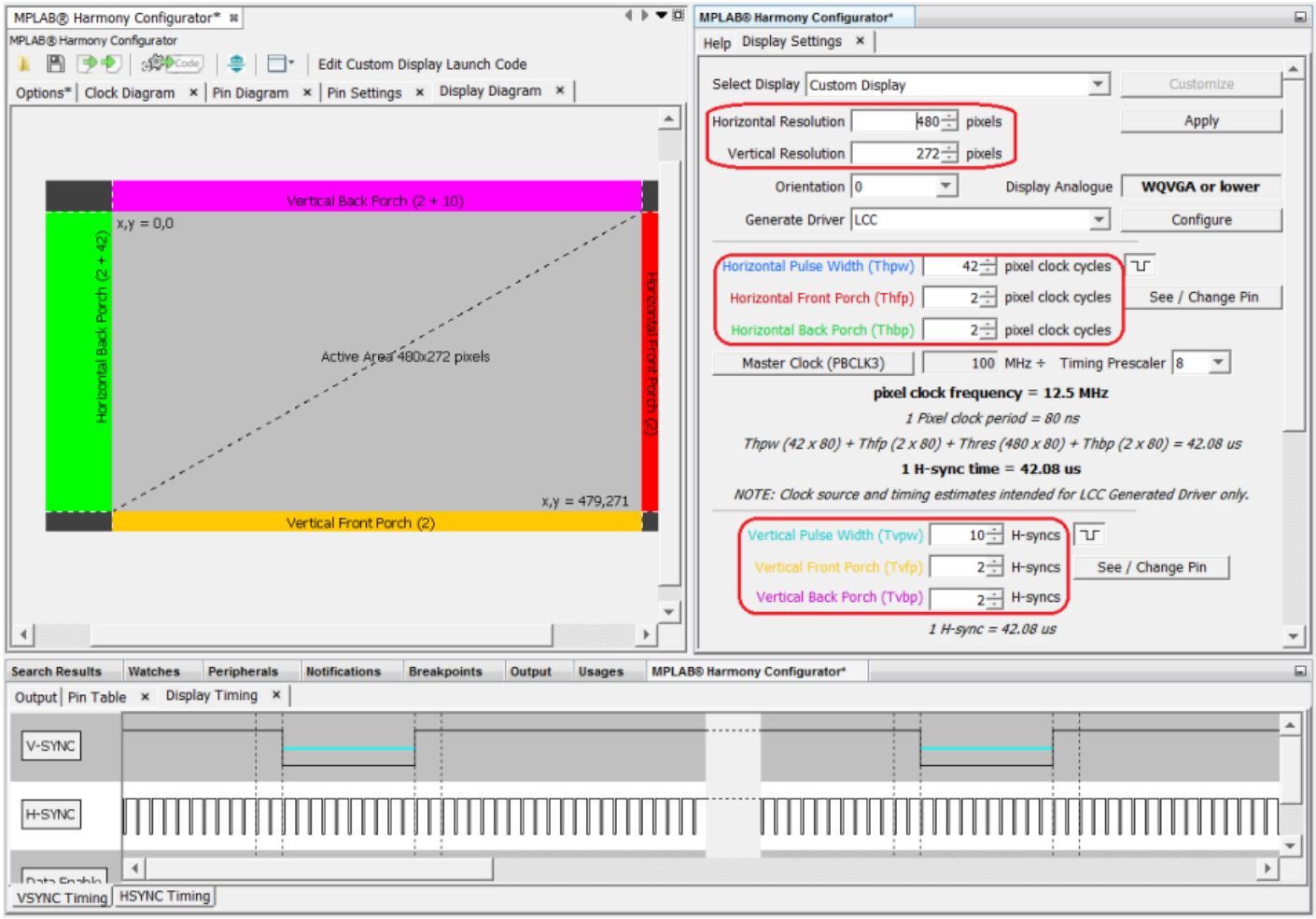


6. Based on the settings of the BSP, the Display Manager will default to the Newhaven 4.3-inch 480x272 WQVGA display. Select **Customize** in the Display Settings tab to enable the fields for entering custom display timing values.

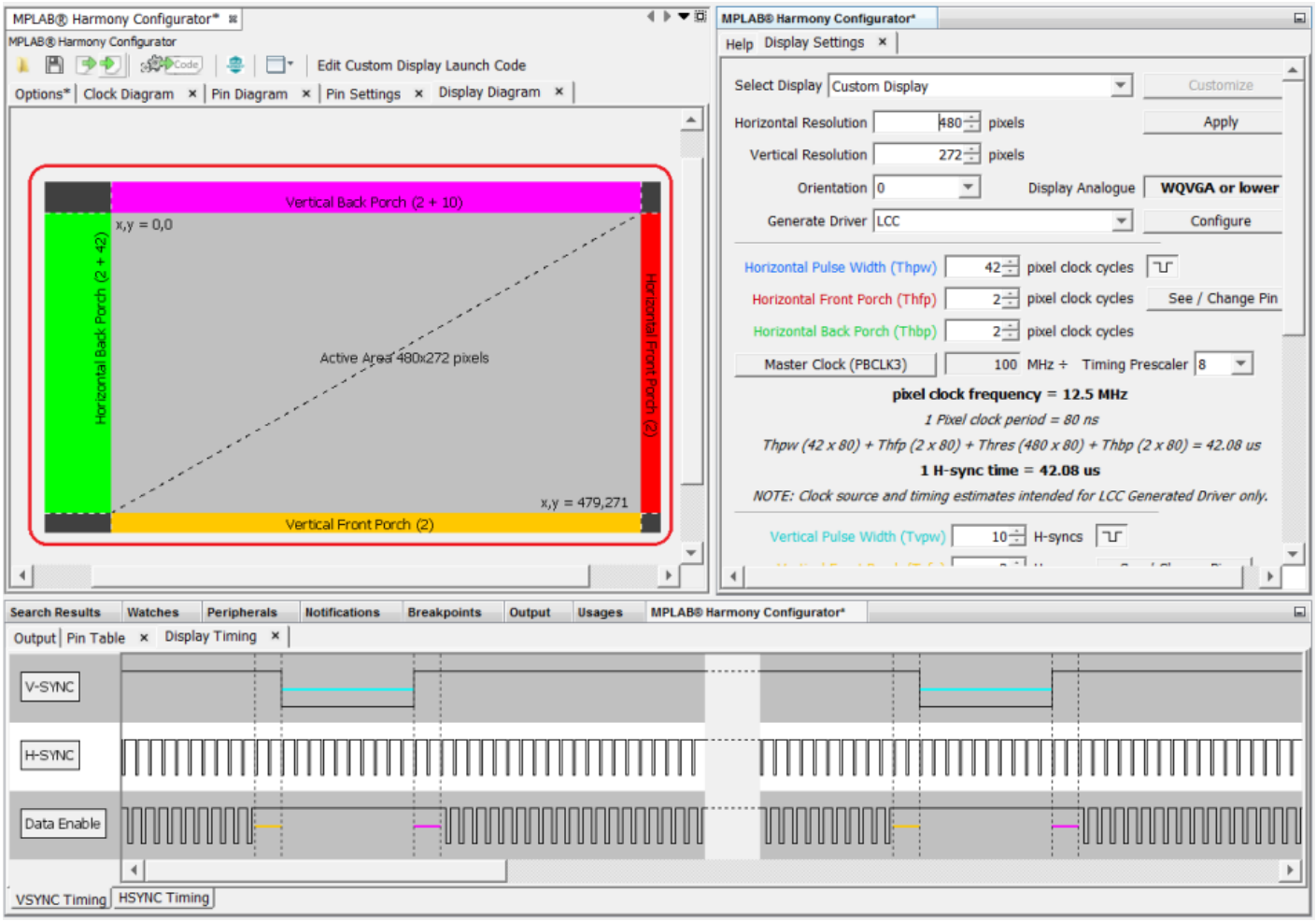
Note: If you choose to use the Newhaven display. certain specification values from the manufacturer's data sheet are required during the configuration process in the MPLAB Harmony Display Manager. A PDF of this data sheet can be obtained from Newhaven Display International, Inc. at: http://www.newhavendisplay.com/specs/NHD-4.3-480272EF-ATXL-CTP.pdf.



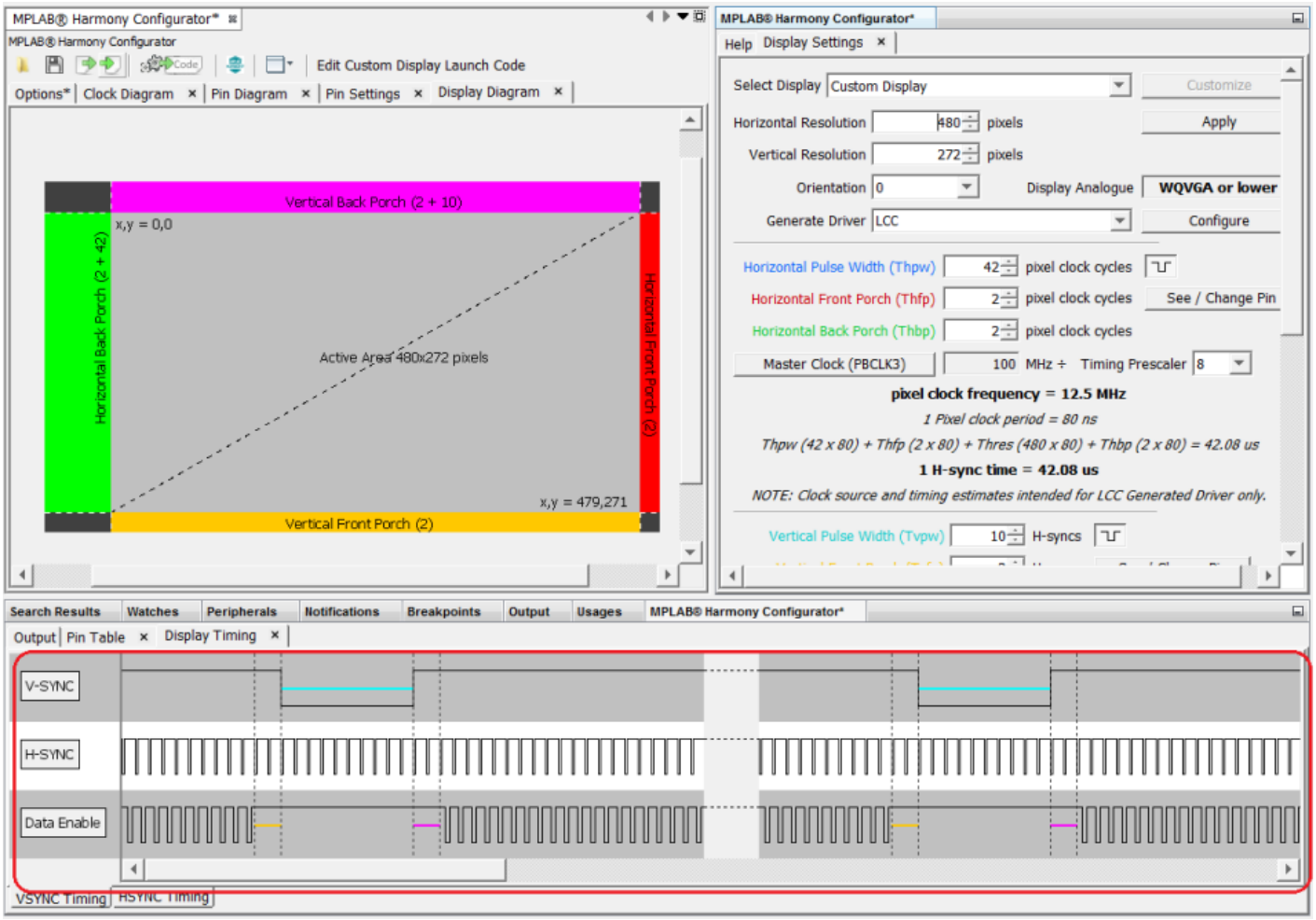
7. In the display data sheet, locate the following values and enter them into their respective fields in the Display Settings tab:
   - Horizontal Pixel Resolution
   - Vertical Pixel Resolution
   - Horizontal Pulse Width (Typically listed as T_{hpw} in the data sheet)
   - Horizontal Front Porch (Typically listed as T_{hfp} in the data sheet)
   - Horizontal Back Porch (Typically listed as T_{hbp} in the data sheet)
   - Vertical Pulse Width (Typically listed as T_{vpw} in the data sheet)
   - Vertical Front Porch (Typically listed as T_{vfp} in the data sheet)
   - Vertical Back Porch (Typically listed as T_{vbp} in the data sheet)



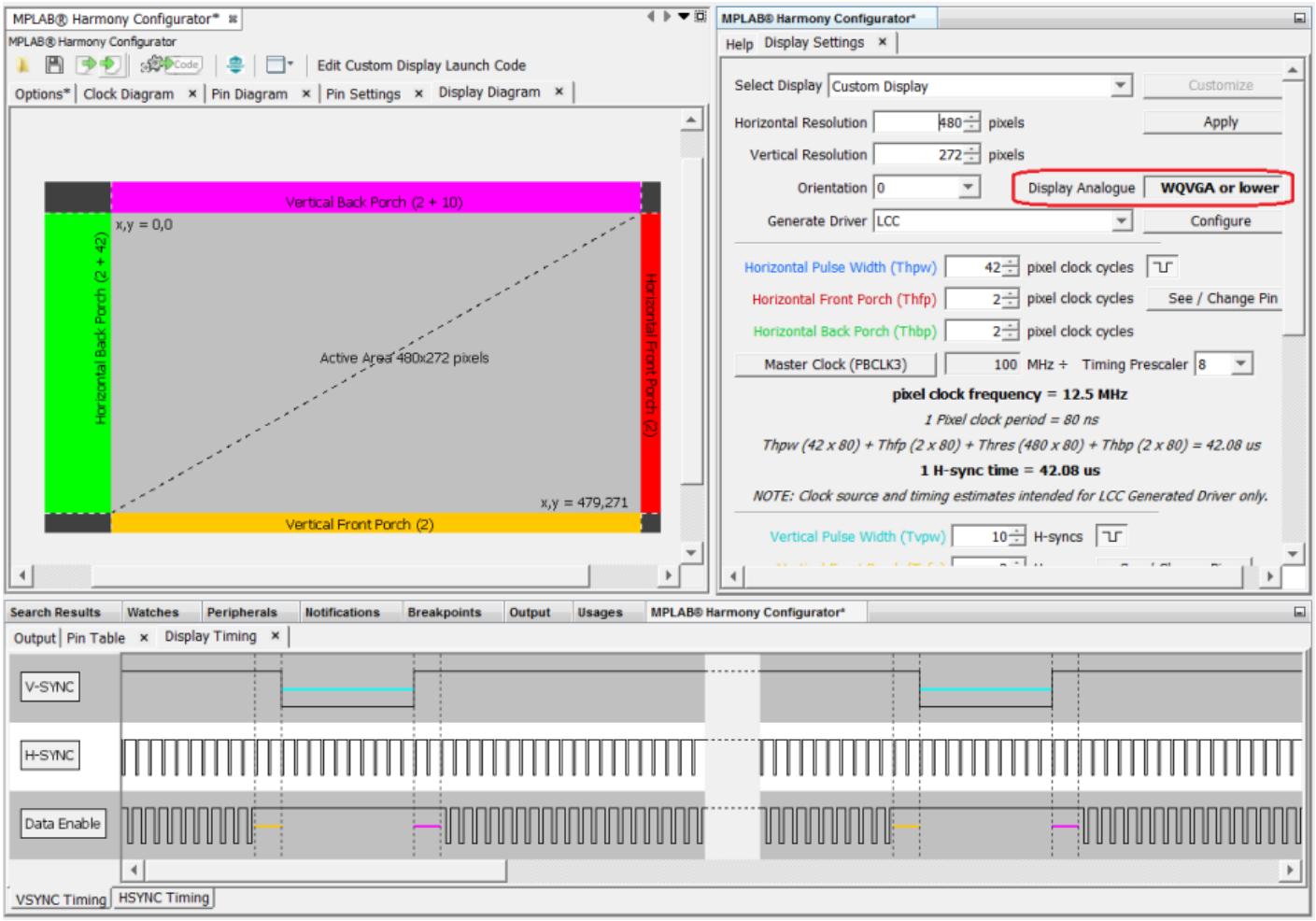
8. The display data sheet may include a diagram depicting the active area transposed over the hidden area. The Display Diagram tab is intended to simulate this diagram based on the values entered in step 7. You may want to visually compare the two.



9. Typically, most display data sheets include waveform diagram depicting the timing interaction between the pixel clock (P-clock) signal, Vertical Sync (V-sync) signal, the Horizontal Sync (H-Sync) signal, and the Data Enable (DE) signal. Some displays may not require a Data Enable signal. The Display Timing tab shows a timing simulation of how the graphics controller will behave. You may want to compare the simulation with the waveform diagram in the display data sheet.

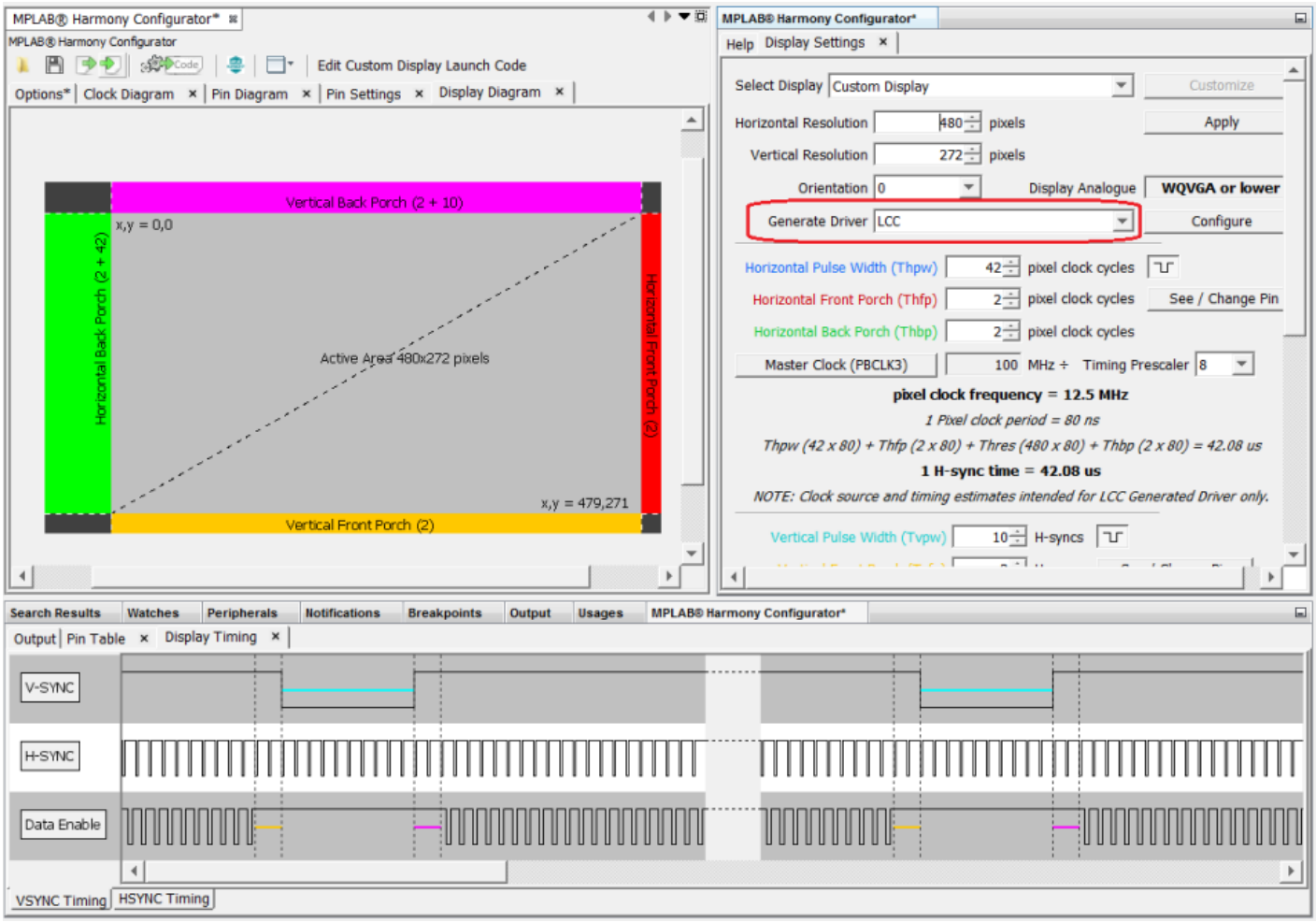


10. Based on the total number of pixels, the Display Manager will estimate a rough analogy of standard display resolution. The estimate is shown in the Display Analogue field in the Display Settings tab. If the display resolution is estimated by the Display Manager to be greater than the largest resolution supported, this field will show “Not Supported”.

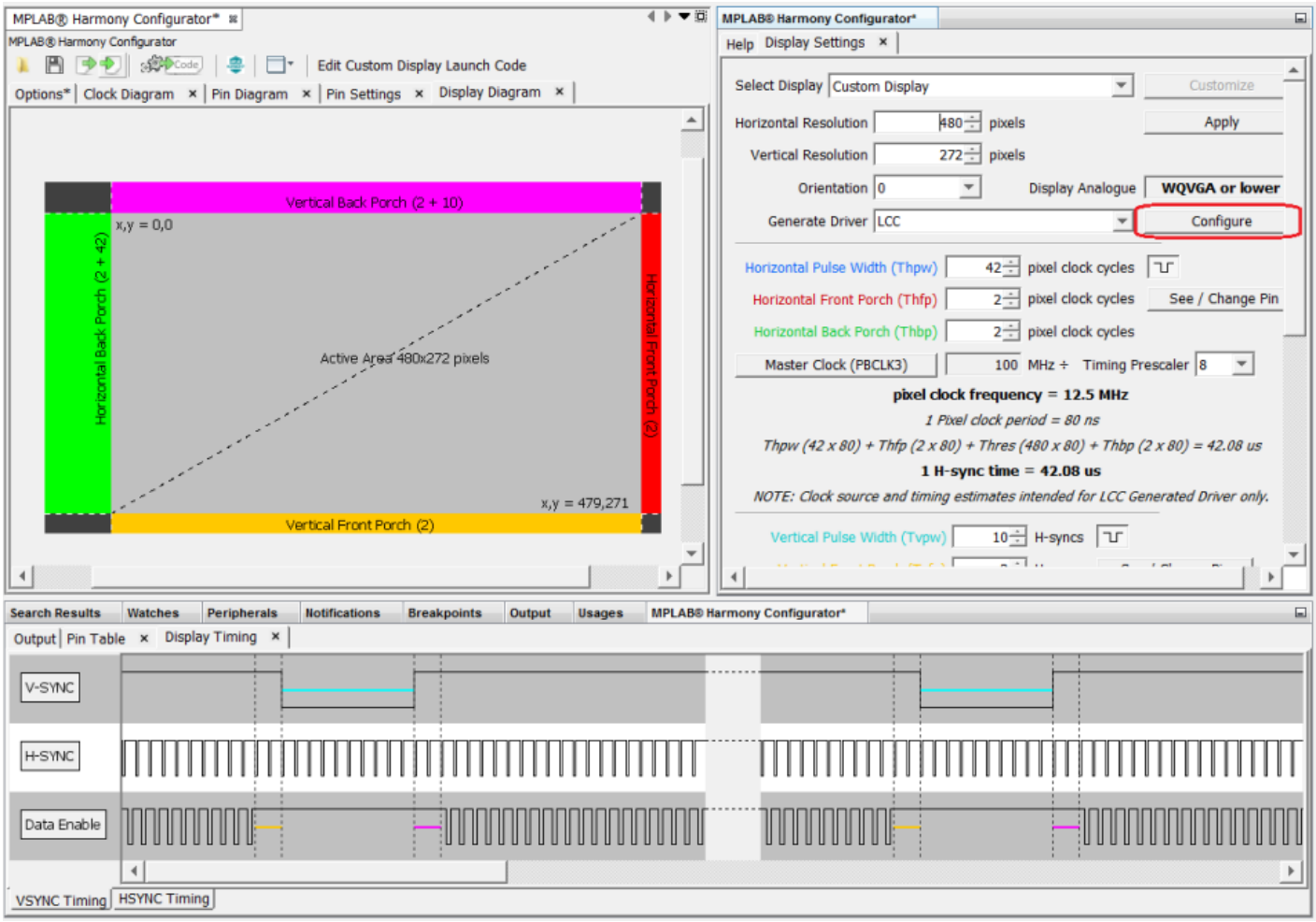


11. Since we are using the MEB II development board, the Low-Cost Controllerless (LCC) Graphics Controller will be used. The Display Manager has an option to provide a generated driver custom tailored for your display. The MEB II BSP should have been preselected to generate LCC . If not, select **LCC** from the Generate Driver drop-down in the Display Settings tab. This will expose more configuration features within the Display Manager specific to the LCC Controller Driver. For more information regarding the LCC technology, please refer to application note AN1387 *"Using PIC32 MCUs to Develop Low-Cost Controllerless (LCC) Graphics Solutions*", which is available for download from the Microchip web site (www.microchip.com).

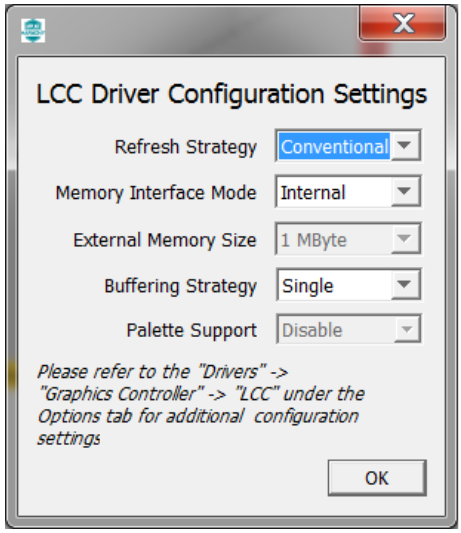
Note: The timing values entered in Step 7 still applies to the graphics controller even if you are not using the LCC Controller Driver.



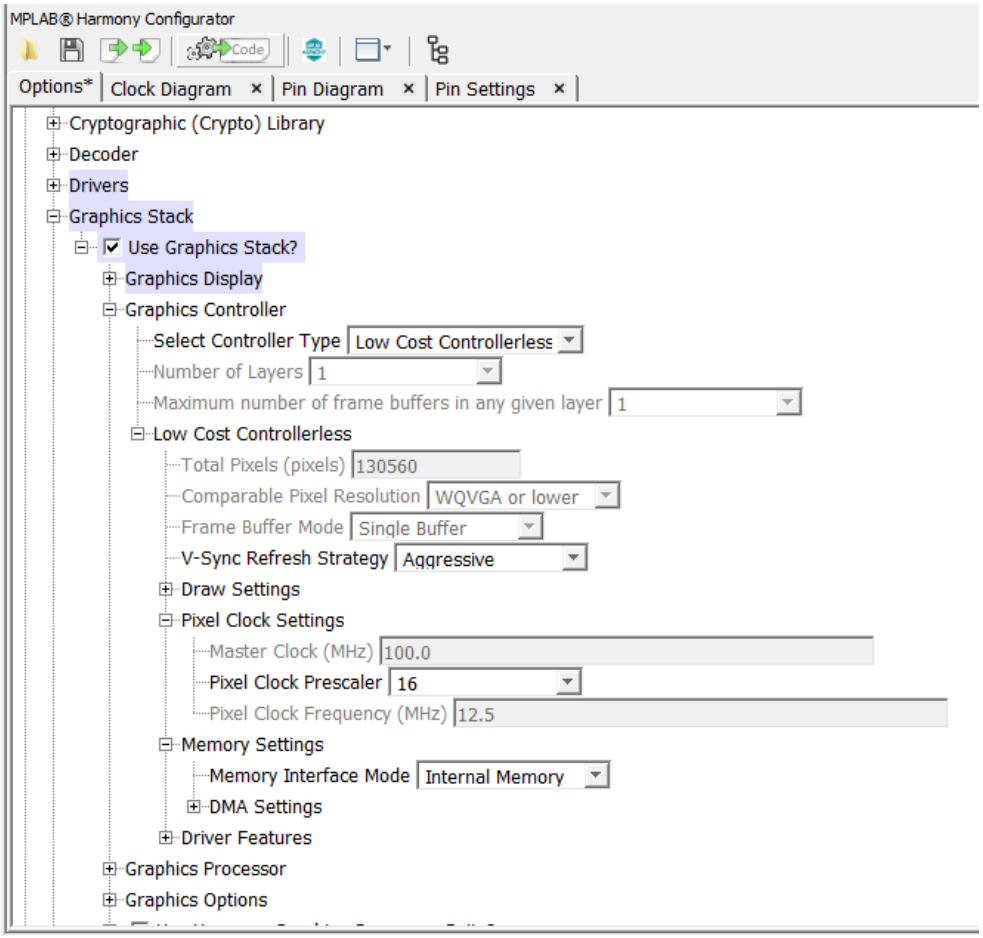
12. Select **Configure** in the Display Settings tab to open the LCC Configuration user interface.



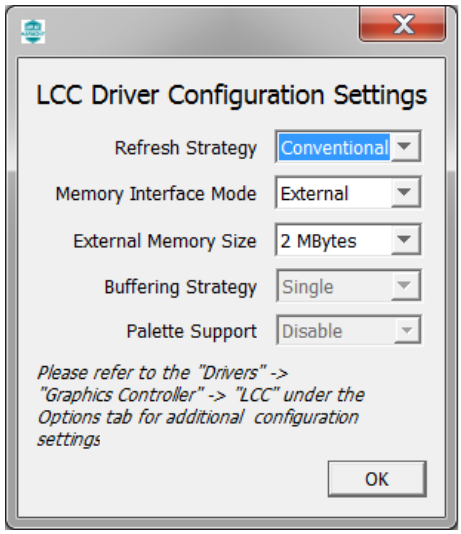
13. The LCC Driver Configuration Settings user interface contains several key settings for quickly setting up the LCC Controller Driver. For now, we will select **Conventional** under Refresh Strategy, as this tells the LCC Controller Driver to use the refresh algorithm most likely to allow the display to render.



Note: The LCC Controller Driver contains more configuration settings under the MPLAB Harmony & Application Configuration tree in the Options tab of the MPLAB Harmony Configurator (MHC). The LCC Driver Configuration Settings user interface services the essential configuration settings. For further system fine-tuning, you may wish to inspect the configuration settings within the Options tab of the MHC.



14. The next field to consider is the Memory Interface Mode field. This field directs the LCC Controller Driver, for its display frame buffer, whether to use the PIC32's internal memory or an external memory via the External Bus Interface (EBI).



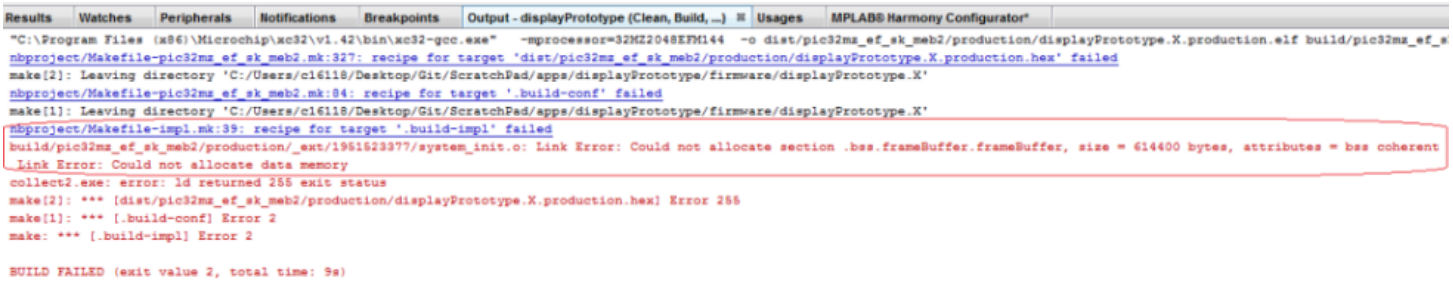
Depending on the display size, you may need to utilize the 2 MB SRAM on the MEB II to accommodate the necessary frame buffer (and in the case of double buffering, two times that amount). The following table can be used to guide you with the selection. Please note that information is provided based on the assumption of 512K memory being internally available on the PIC32MZ device.

Display Analogue	Buffering Strategy	Memory Interface Mode
WQVGA or lower	Single	Internal
WQVGA or lower	Double	External
HVGA	Single	Internal
HVGA	Double	External
VGA	Single	External

VGA	Double	External
WVGA	Single	External
WVGA	Double	External

The Display Manager will provide you with a warning pop-up if it detects the current configuration may not have enough memory to support the display resolution.

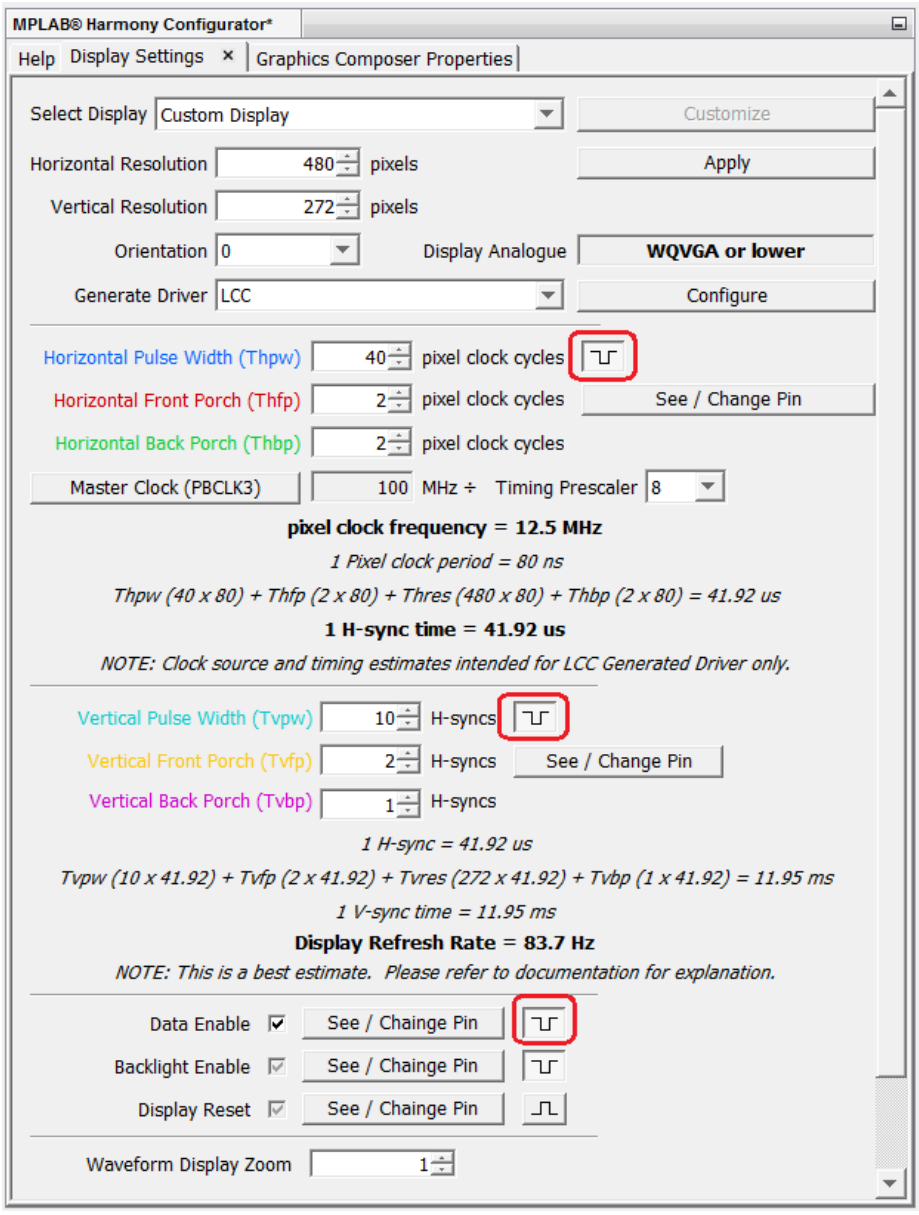
Another checking mechanism, in the case of frame buffer being too large for the supported memory, is a compile error such, as shown in the following figure.



Note: Make sure to select **2 Mbytes** when using the external memory, as it is the size of the SRAM connected to the EBI on the MEB II.

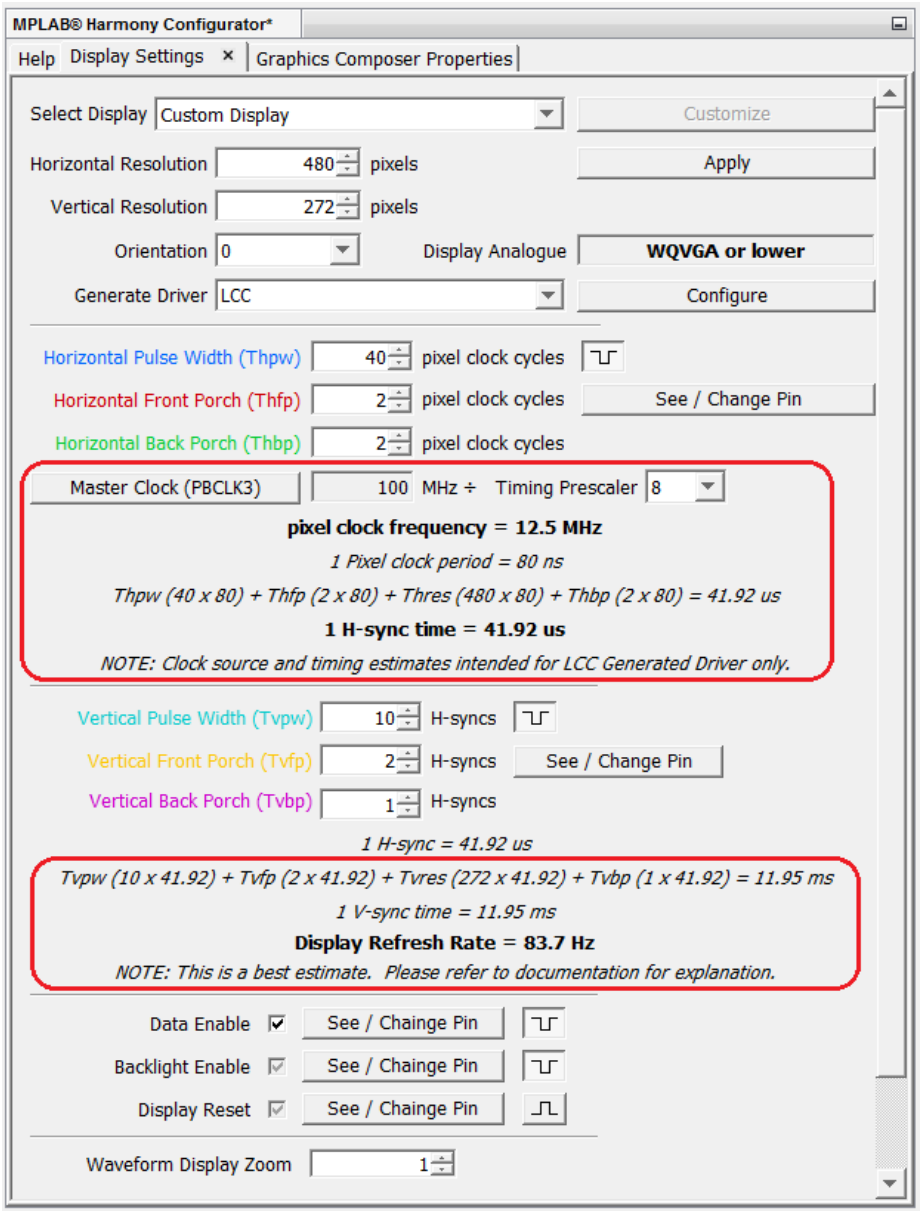
15. Depending on your display, the polarity of the V-Sync, H-Sync, and DE pulses can be inverted, respectively. You will want to do so to match the simulated waveform with the data sheet waveform.

Note: Polarity control of the pulses is only available with the LCC generated driver for the current release.

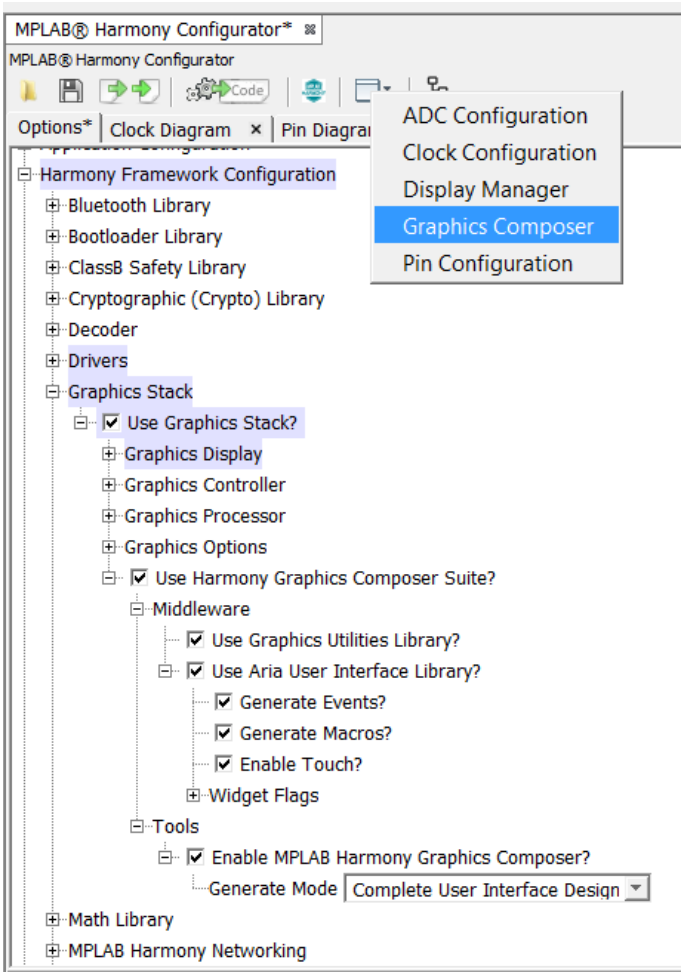


16. If the LCC generated driver is used, the Display Manager also provides an estimated refresh rate base on the master clock source and timing values provided. The estimated timing will be updated as the timing values are adjusted. The important number to note is the Display Refresh Rate. You may want to ensure this number is within the tolerance specification in the display data sheet. There are two ways to adjust the clock source:
    - The first method is selecting **Master Clock** to open the Clock Configurator. Depending on your PIC32 device, the button will indicate which clock to adjust. In the case of the PIC32MZ, the clock source to adjust is PBCLK3.
    - The second method is to adjust the LCC generated driver's internal prescaler setting. Admittedly, this is a coarser adjustment setting

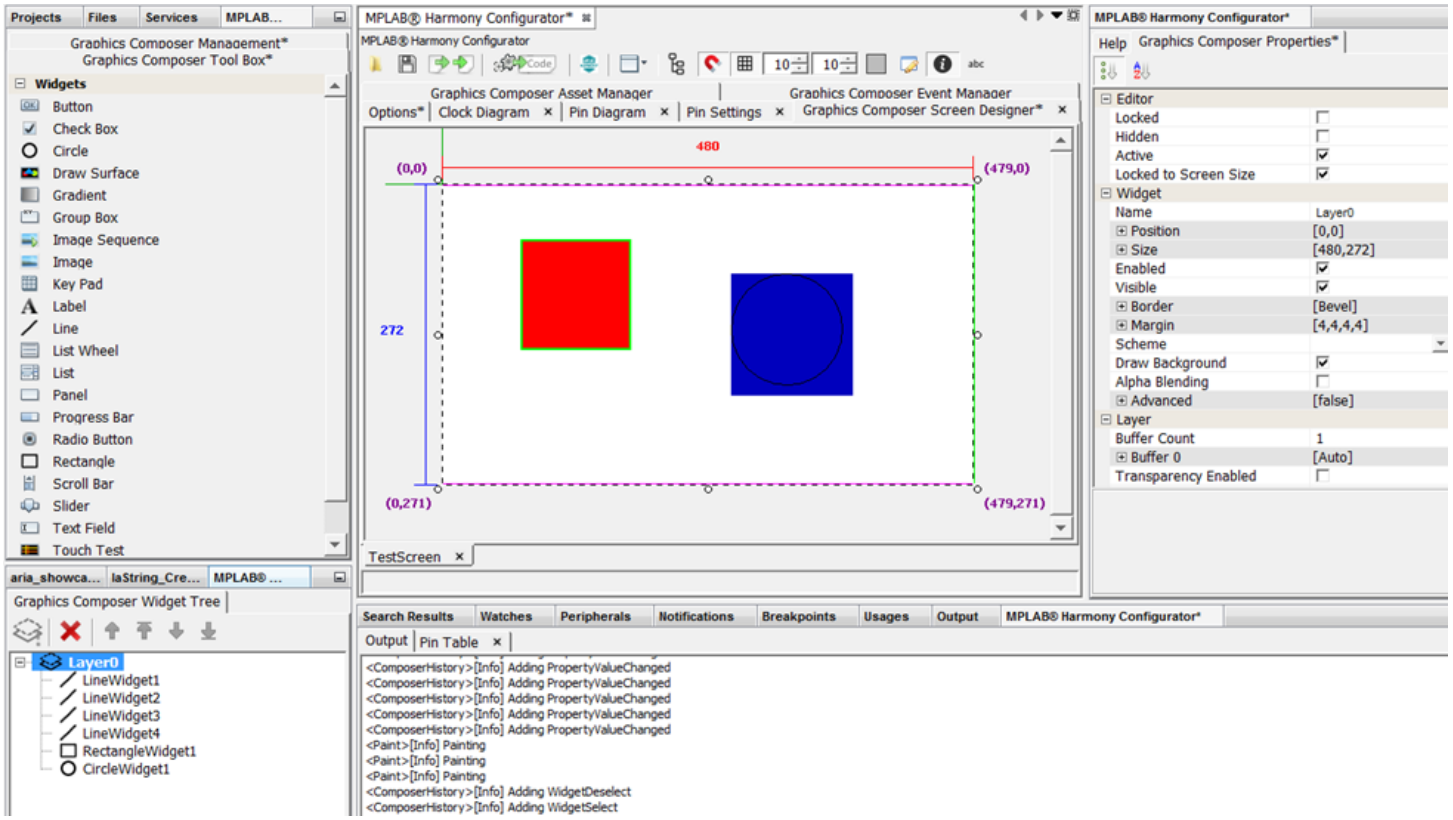
Note: Typical estimated timing tends to be approximate 5 to 10% higher in frequency than the actual measured value, as the pixel clock for the LCC driver is driven by the clocking of each byte on the DMA channel, the DMA channel is preempted every time the Graphics Library is updating the frame buffer in a draw event. This pause is what slows down the entire refresh rate.



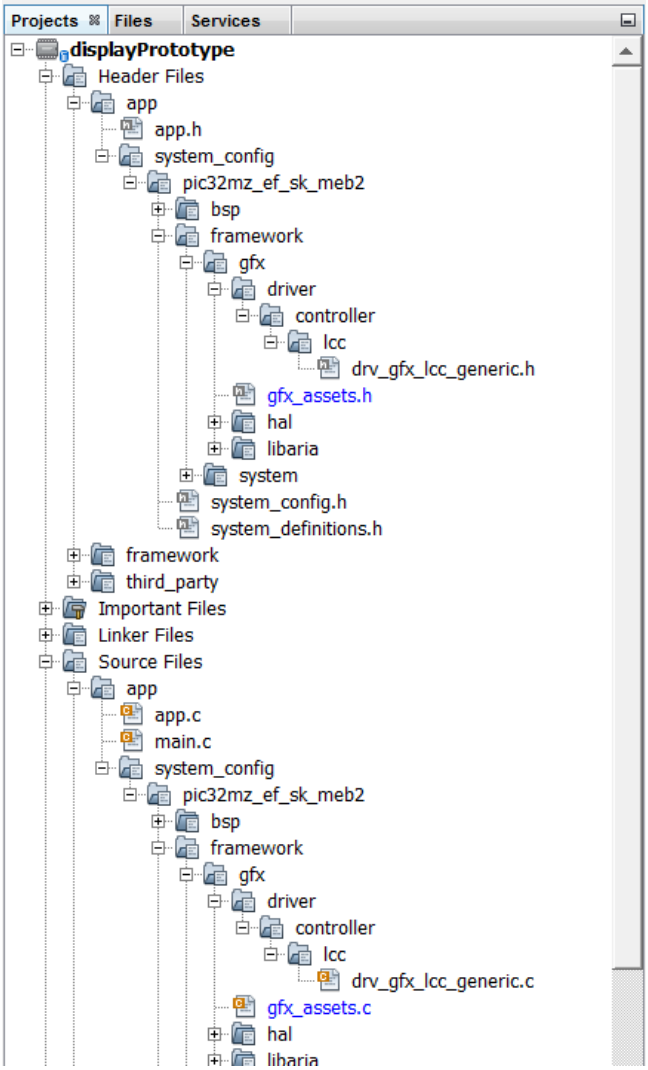
17. Next, we will enable the MPLAB Harmony Graphics Composer (MHGC) to create a test screen for the display. For details about the MHGC, please refer to the MPLAB Harmony Graphics Composer User's Guide.



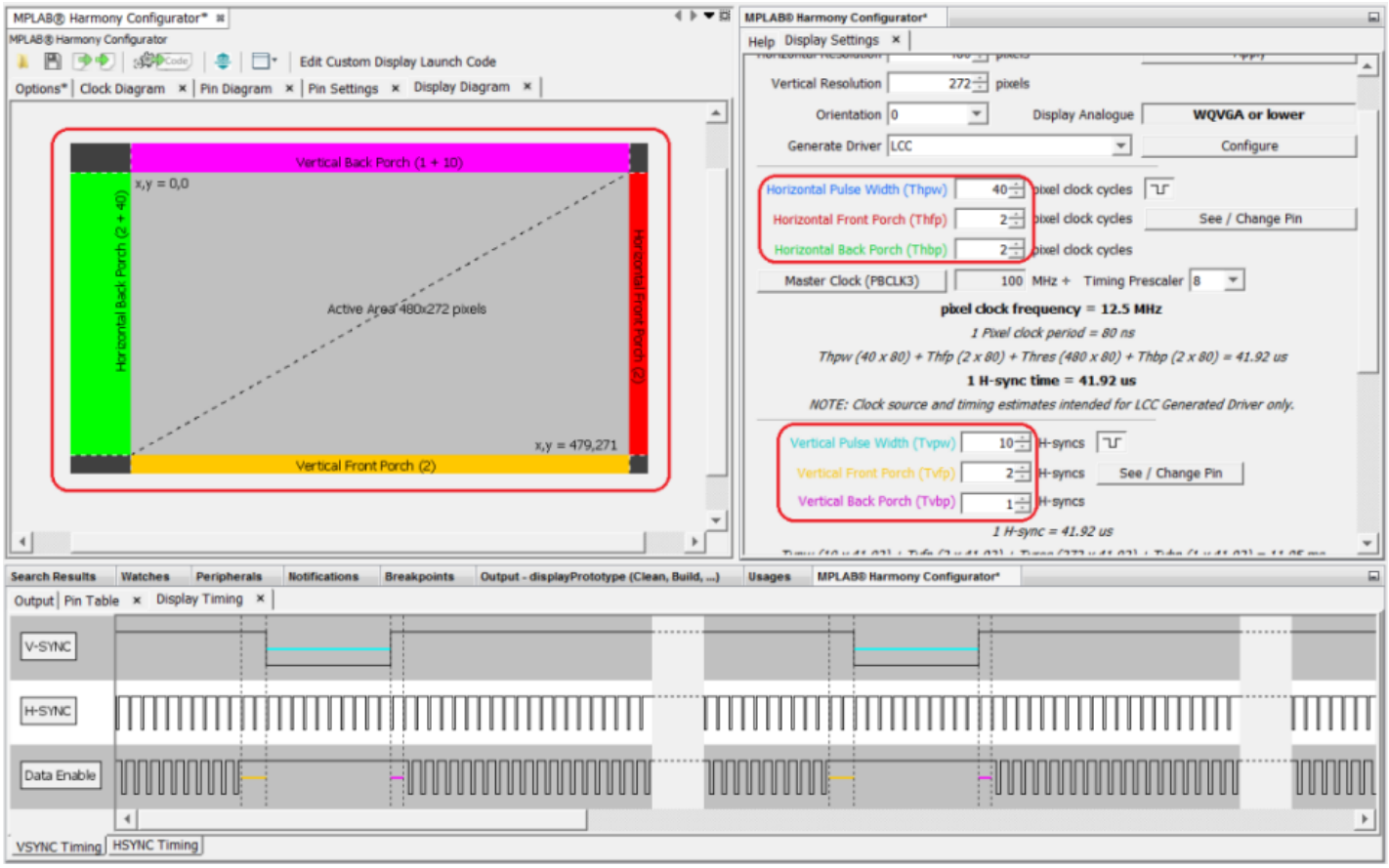
18. Using the MHGC, draw a test screen with a white background, and four line widgets along the border of the screen, as shown in the following figure. You can use the Snap button to get the lines right up against the edge. You may want to add a couple of primitive shapes with fill color so that you can easily see if the display is rendering the graphics properly. The test screen is designed to ensure the timing values entered in step 7 are adequate for the display.



19. The project is now ready to be generated and deployed to the MEB II. During generation, the Display Manager will add the generated LCC driver files `drv_gfx_lcc_generic.h` and `drv_gfx_lcc_generic.c` to `app/system_config/<configuration name>/framework/driver/gfx/controller/lcc`.



20. Once deployed, you should see the test screen rendered and the single pixel-width color lines in the test screen should be rendered right at the edge of the screen. If you do not see the lines rendered at the edge or they are only partially rendered, you may need to tune the timing values using the Settings Tab. Use the Active Area in the Display Diagram tab as your positional reference.



21. You now have a baseline LCC driver configuration for rendering static graphics on your display. However, the driver may not handle image decoding, motion, or screen transition well. Enabling Double Buffering or adjusting the Display Refresh Rate are two ways to improve display driver performance.

The process is now complete; however, you may want to revisit the Display Manager later and use it as a display tuning tool by allowing a short adjustment-to-deploy feedback cycle.

MPLAB Harmony ADC Manager User's Guide

This topic provides user information about using the MPLAB Harmony ADC Manager.

Introduction

The MPLAB Harmony ADC Manager is a design tool that is integrated as part of the MPLAB Harmony Configurator (MHC). This tool allows a user to easily configure the ADC on PIC32MZ EF, PIC32MZ DA, and PIC32MK families of devices.

Description

- The overall development flow of ADC Manager consists of:
- General clock settings
- Reference Voltage settings
- Individual ADC settings, which include:
  - Input selection, including single-ended or differential mode
  - Resolution selection
  - Sampling Rate with Auto-Calculate feature
  - Trigger source selection
  - Interrupt or Polling selection
- Enabling individual inputs for the shared ADC
- Source selection for scan trigger

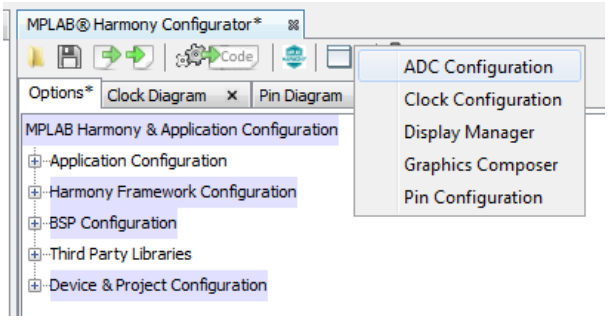
Getting Started

This section describes how to begin using the MPLAB Harmony ADC Manager.

Description

To begin using the ADC Manager, which is part of the MPLAB Harmony Configurator (MHC), you will need to create a new MPLAB Harmony project and select a PIC32 device that has the High-speed ADC (ADCHS) feature. These devices include the PIC32MZ EF, PIC32MZ DA, and PIC32MK family of devices. For example, your project could be named adchs_demo. Once you have created your project, do the following:

1. Open MPLAB Harmony Configurator.
2. From the **Launch Utility** button on the menu bar, select **ADC Configuration**.



User Interface

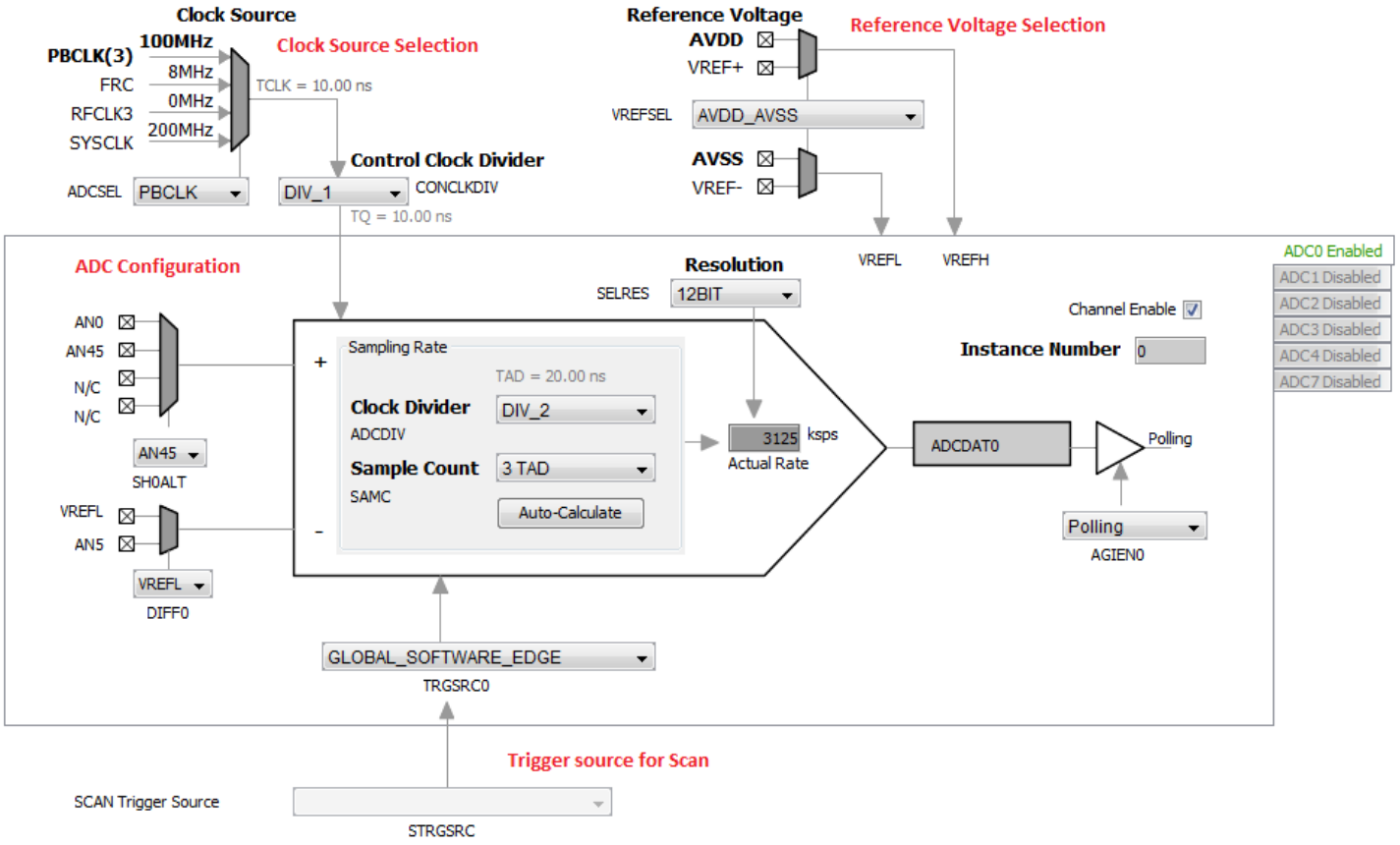
This section describes the elements of the user interface for the MPLAB Harmony ADC Manager.

Description

The following describes each section of the MPLAB Harmony ADC Manager user interface.

User Interface Layout

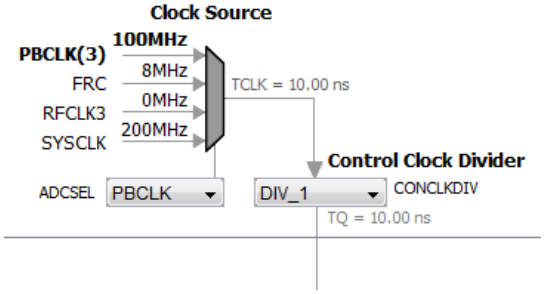
The following figure shows the initial user interface layout.



Clock Source Selection

The Clock Source Selection area:

- Is the interface for choosing a clock source for all of the ADC modules, and how that clock source is divided for the final ADC clock (TQ).
- Lets you configure the source and speed of the overall ADC clock, which affects all ADC modules.



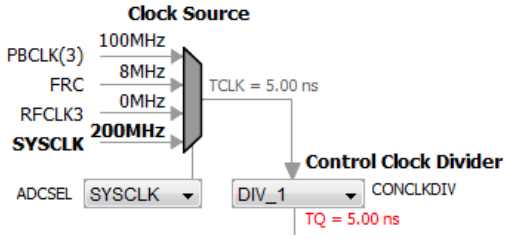
The High-speed ADC has four possible clock sources:

- The Peripheral Bus Clock (PBCLK)
- The Internal 8 MHz Oscillator (FRC)
- One of the Reference Clock Modules (RFCLKx)
- The System Clock (SYSCLK). On some parts, the SYSCLK will automatically be divided by two before getting to the ADC clock divider.

The clock source is selected with the drop-down menu on the left. If the Reference Clock is selected, it must be configured using the Clock Manager screen. Please refer to the Clock Manager help documentation for information about how to configure the Reference Clock.

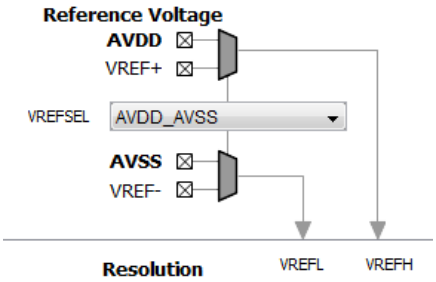
The drop-down box on the right sets a clock divider between the source clock and the ADC modules.

Note: The ADC has a requirement for a minimum time for TQ. If the speed of the clock going into the ADC is faster than the requirement, the label with “TQ =” will turn red to show the violation of the timing specification. To resolve it, select another clock source, a different divider, or adjust the clock settings by using the Clock Manager.



Reference Voltage Selection

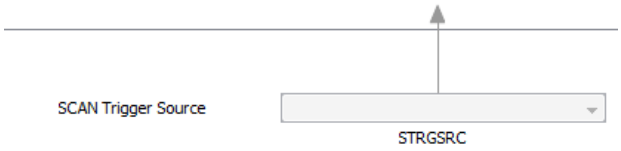
The Reference Voltage Selection area lets you configure the reference voltages (VREFL and VREFH) that are used for all ADC modules.



The options available depend on the device in use. When you make a selection from the drop-down menu, the labels (AVDD/VREF+, AVSS/VREF-) change the highlight according to the selection.

Scan Trigger Source Selection

This drop-down menu lets you configure the interrupt source that is used when scanning is triggered. This is done selecting which interrupt source will trigger a scan operation. Selecting this interrupt source mainly affects the shared ADC (ADC7); however, the dedicated channels can also use the same source if desired.



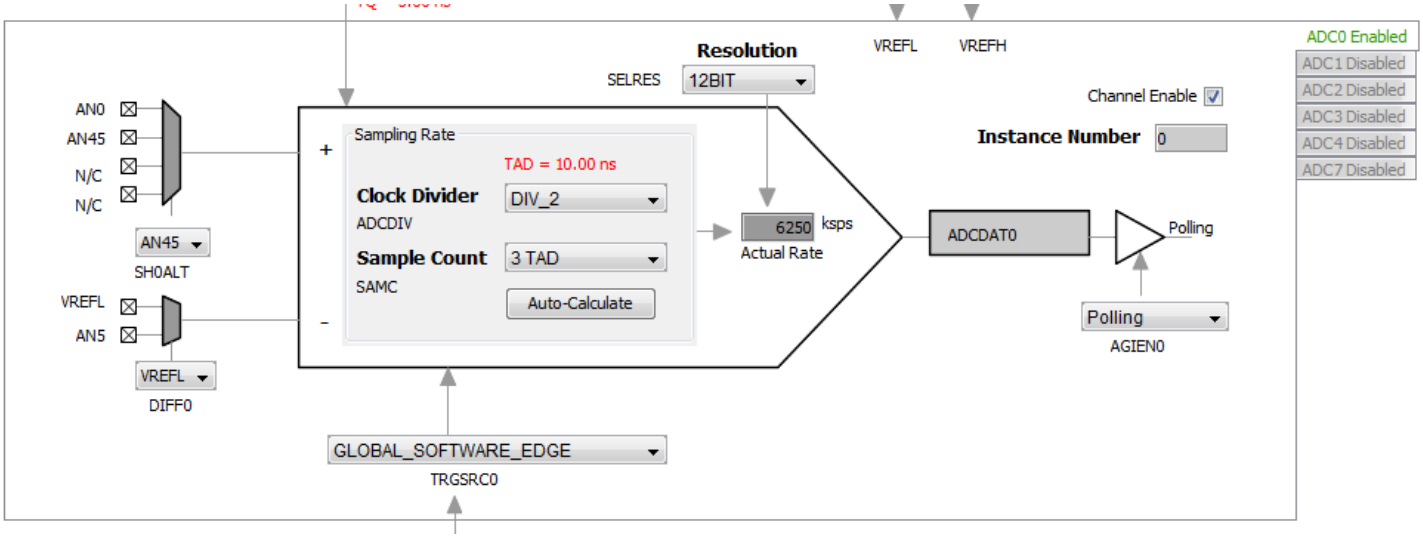
The options for the Scan Trigger Source depends on the part in use. Most of the options are the same between all devices, such as the Global Software Level trigger and the Global Software Edge trigger. Refer to the device-specific data sheet for information about which trigger sources are available.

Note: This does not configure the trigger source, which must be done separately within MHC. For example, if you have selected a PWM or TMR as the trigger source, that module must be configured separately within MHC.

ADC Configuration

The ADC Configuration section lets you configure each ADC channel, what inputs are in use, whether it is differential or single-ended, along with sampling rate, and whether the data from the ADC results in an interrupt.

Each available ADC module is chosen on the right side of the area. From there, each ADC can be individually configured for input sources, resolution, sampling rate, trigger source for starting the conversion (dedicated channels) or scan mode (shared channels), and whether the module will interrupt the processor when conversions are complete.



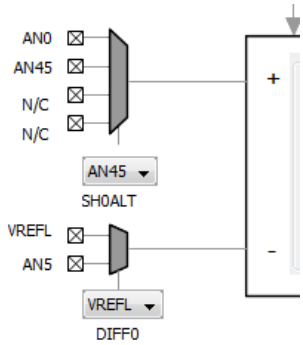
Turning on an ADC channel

When setting up an ADC channel, the first step is to select a particular ADC from the tabs on the right, and then enable the channel by selecting the **Channel Enable** check box.

The **Instance Number** box gives you an idea of which ADC driver instance to refer to when calling various ADC driver functions.

Input Source Selection

The area on the left lets you select which input source is used by the ADC, and whether the channel is single-ended or differential.

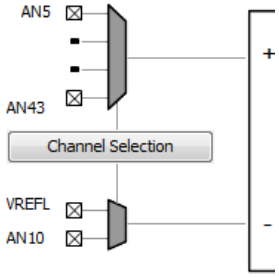


For the dedicated channels, there will be available one alternate source or three alternate sources. Whichever source is selected, the data will arrive the in data register for the first source, that is, the data register for that ADC channel.

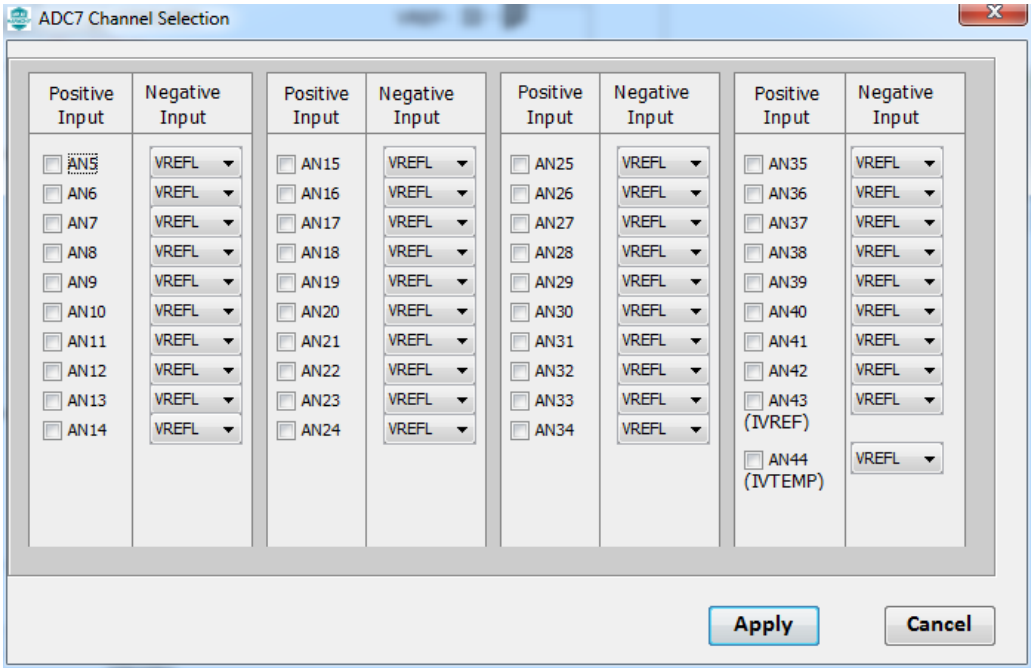
For example, ADC 0 will always write the result into ADCDATA0, irrespective of which input is selected.

The DIFFx drop-down menu lets you select the negative input source for the ADC. If VREFL is selected, the input is configured as single-ended. If the alternate input is selected (AN5 in the figure), the channel is configured as differential.

For the shared channel, there is a different way to select the inputs that are sampled and converted during a scan sequence.



Note: There is no drop-down menu for the negative input to the ADC. Click **Channel Selection**, and a dialog appears with all of the available inputs for the shared ADC.



In the Channel Selection dialog, each available input has a check box to the left, and a drop-down menu to the right. Use the check box to enable the channel input to the shared ADC. You use the drop-down menu to select whether that input is to be in single-ended mode (by using VREFL as the Negative Input), or differential mode (by selecting the alternate input option).

Note: Currently, the dialog shows both Class 2 (independent trigger source) and Class 3 (shared trigger source) inputs; however, selection of the Class 2 trigger sources is not available here. It is necessary at this time to go to the ADC Driver in the regular MHC tree, which is located within *Framework >Drivers >ADC >Enable Analog Input Conversion?*. A future version of the ADC Manager will provide this capability from this dialog.

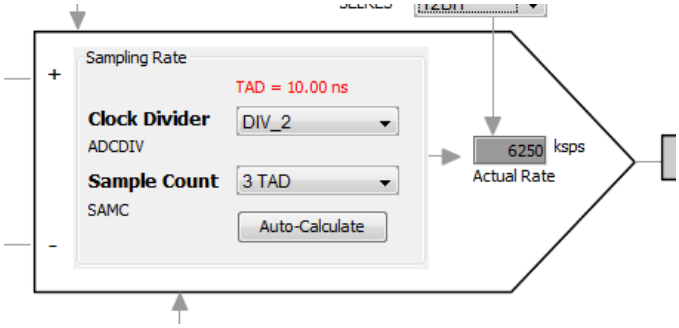
Output Resolution

Each ADC can be configured for different data resolution rates by using the drop-down menu above the ADC symbol.

A lower resolution can yield a faster sampling rate. The available resolutions are 12-bit, 10-bit, 8-bit, and 6-bit.

Individual ADC Sampling Rate selection

Each ADC can be configured for its own ADC clock (TAD) and sampling time (SAMC).

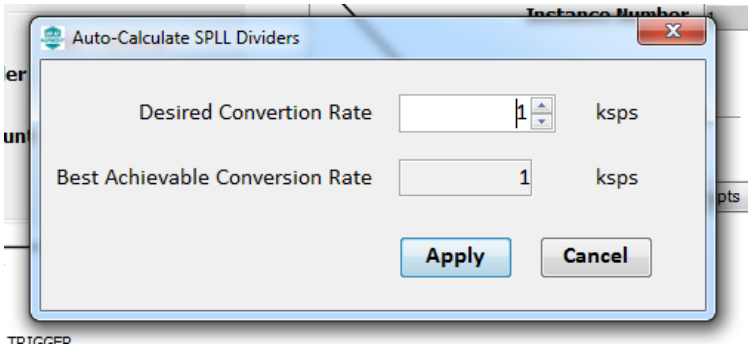


The **Clock Divider** drop-down menu in the **Sampling Rate** panel controls how much the ADC module clock (TQ) is divided to produce the final TAD clock. There are minimum requirements for the TAD timing, and if the current setting violates that minimum, the text label above the drop-down box will turn red.

The **Sample Count** drop-down menu in the **Sampling Rate** panel controls how long the sampling capacitor is connected to the input pin(s) during the sampling phase. For dedicated ADC channels, this controls the minimum time for the sampling, as the ADC goes into sampling phase after a conversion is complete. For the shared ADC, each input that is selected is sampled for the specified number of TAD clocks before going into conversion mode.

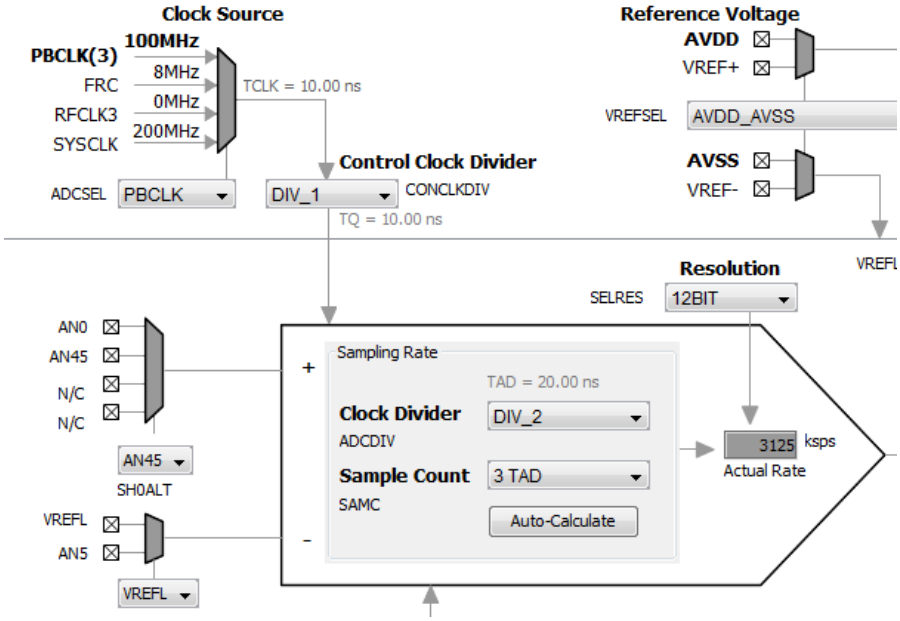
The **Actual Rate** box to the right of the **Sampling Rate** panel indicates how many samples per second will actually be produced by the

combination of Clock Divider and Sample Count. There is also an option for auto-calculating both the ADCDIV and SAMC values based on a desired sampling rate. Clicking the **Auto-Calculate** button displays the following dialog box:

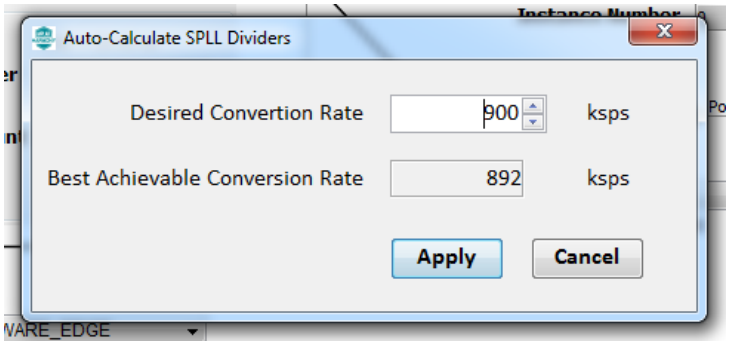


Selecting the desired conversion rate (in ksps) by using the **Desired Conversion Rate** spin buttons shows the achievable conversion rate in **Best Achievable Conversion Rate** box. When the desired setting has been reached, clicking **Apply** updates the new settings to the ADC settings.

As an example, use the PIC32MZ EF device, and use PBCLK as the input to the ADC.

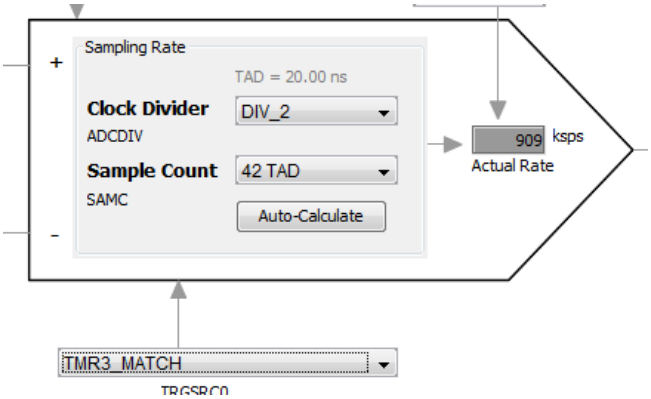


Let's say we would like to have a sample rate of 900 ksps on ADC0. Bringing up the auto-calculate dialog box and entering 900 into the input box, we see that the ADC Manager has determined that the best achievable rate is actually 892 ksps.



Going to 901 ksps provides a best conversion rate of 909 ksps. Therefore, it would be up to the user to decide whether the available rate is acceptable, or whether it would be necessary to adjust clocks or use another clock source.

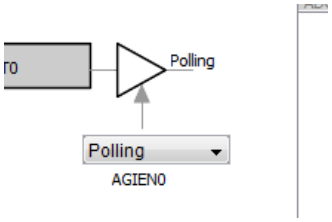
Note: The conversion rate is based on the assumption of continuous sampling and converting, while trigger sources would determine what is actually accomplished. Therefore, it might be necessary to set the ADC for a faster rate than the desired rate (i.e., the 909 ksps), and then adjust the trigger source (e.g., TMR3 match) to get the exact conversion rate desired. If the 909 ksps is accepted, we see the ADC configured as follows:



In this case, the TQ clock is divided by 2 for a TAD clock of 20 ns (50 MHz). The Sample Count (SAMC) is set for 42 TAD, so the sample phase will take 840 ns (20 ns * 42).

Polling/Interrupt Selection

Each ADC can be configured to generate an interrupt when a sample has been converted. For the dedicated channels, use the drop-down menu to the left of the ADC symbol to make the selection.



Note: The shared ADC channel has a button that displays a dialog box for selecting **Interrupt** for **Polling** for each of the available inputs to the ADC. However, this dialog box does not modify anything in the MHC tree, so it is not necessary to use it at this time.

Conclusion

Using the ADC Manager can allow setup of the ADCs in PIC32MZ EF, PIC32MZ DA, and PIC32MK devices to be simple and elegant.

MPLAB Harmony Graphics Composer User's Guide

This section provides user information about using the MPLAB Harmony Graphics Composer (MHGC).

Introduction

This user's guide provides information on the MPLAB Harmony Graphics Composer (MHGC), also referred to as the graphics composer, which is included in your installation of MPLAB Harmony. MHGC is tightly coupled with the Aria User Interface Library to facilitate rapid prototyping and optimization of the application's graphical user interface (GUI).

Description

The MPLAB Harmony Graphics Composer (MHGC), also referred to as the graphics composer, is a graphics user interface design tool that is integrated as part of the MPLAB Harmony Configurator (MHC). MHGC is tightly coupled with the Aria User Interface Library to facilitate rapid prototyping and optimization of the application's graphical user interface (GUI). The tool provides a "What you see is what you get" (WSYWIG) environment for users to design the graphics user interface for their application. Refer to *Volume V: MPLAB Harmony Framework Reference > Graphics Library Help > Aria User Interface Library* for more information.

The MPLAB Harmony Graphics Composer (MHGC) Tool Suite and the Aria User Interface Library provide the following benefits to developers:

- Enhanced User Experience – Libraries and tools are easy to learn and use.
- Intuitive MHGC Window Tool – Flexible window docking/undocking. Undo/Redo and Copy/Paste support. Tree-based design model. Display design canvas control including zooming.
- Tight Integration Experience – Graphics design & code generator tools are tightly integrated, providing rapid prototyping and optimization of look and feel
- Powerful User Interface (UI) Library – Provides graphics objects and touch support
- Multi-Layer UI design – Supported in the MHGC tool and Aria Library
- Complete Code Generation – Can generate code for library initialization, library management, touch integration, color schemes and event handling with a single click
- Supports Performance and Resource Optimization – Draw order, background caching, and advanced color mode support improve performance
- Resource optimization – Measures Flash memory usage and can direct resources to external memory if needed. Global 8-bit color look-up table (LUT) supports reduced memory footprint. Heap Estimator tool, which helps to manage the SRAM memory footprint.
- Text localization – Easily integrate international language characters into a design and seamlessly change between defined languages at run-time
- Easy to Use Asset Management – Tools provide intuitive management of all graphics assets (fonts, images, text strings)
- Image Optimization – Supports cropping, resizing, and color mode tuning of images
- Expanded Color Mode Support – The graphics stack can manage frame buffers using 8-bit to 32-bit color
- Powerful Asset Converter – Inputs several image formats, auto converts from input format to several popular internal asset formats, performs auto palette generation for image compression, supports run-length encoding. Supports automatic font character inclusion & rasterization.
- Event Management – Wizard-based event configuration. Tight coupling to enable touch user events and external logical events to change the graphics state machine and graphics properties.
- Abstract Hardware Support – Graphics controllers and accelerators can be added or removed without any change to the application

Glossary of Terms

Throughout this user's guide the following terms are used:

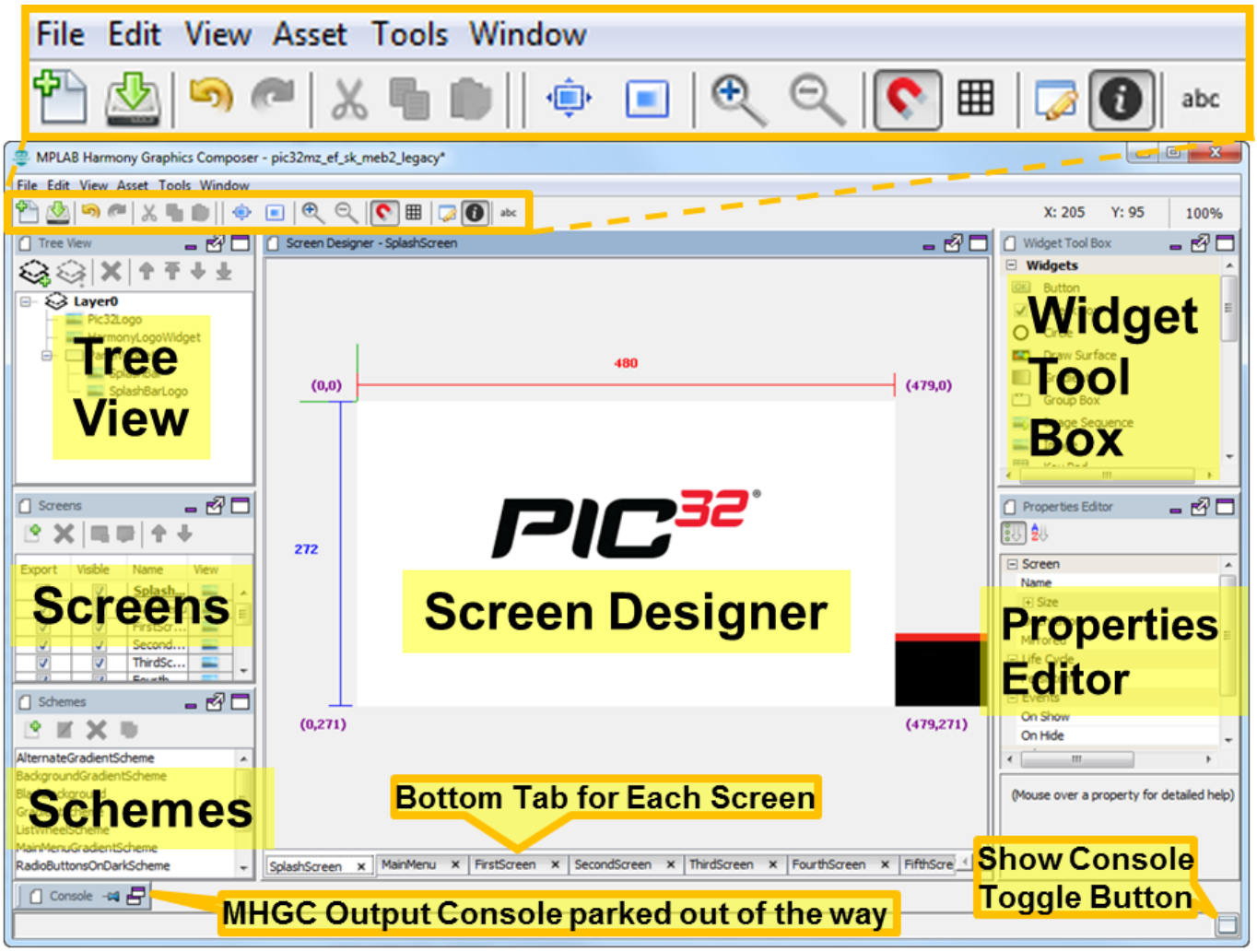
Acronym or Term	Description
Action	A specific task to perform when an event occurs.
Asset	An image, font, or binary data blob that is used by a user interface.
Event	A notification that a specific occurrence has taken place.
Resolution	The size of the target device screen in pixels.
Screen	A discreet presentation of organized objects.
Tool	An interface used to create objects.
UI	Abbreviation for User Interface.
Widget	A graphical object that resides on the user interface screen.

Graphics Composer Window User Interface

This section describes the layout of the different windows and tool panels available through MHGC.

Description

MHGC is launched from the MHC toolbar Launch Utility menu. Launching the Graphics Composer creates a new screen. Shown below is the MHGC screen for the Aria Showcase demonstration. (If you don't see this screen layout, reset the screen by selecting *Window* > *Reset Dock Areas* from the window's menus.)



Panels

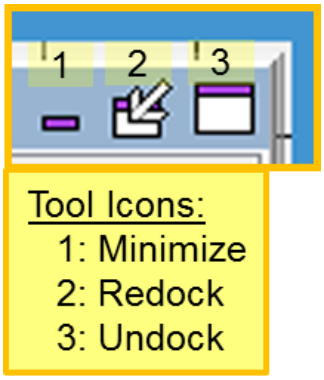
By default, there are five active panels and one minimize panel on this screen:

- Screen Designer – Shows the screen design for the selected screen. Tabs on the bottom of the Screen Designer panel show the available screens.
- Tree View – Shows the layer and widget hierarchy for the current screen.
- Screens – Manages screens in the application.
- Schemes – Manages coloring schemes in the application.

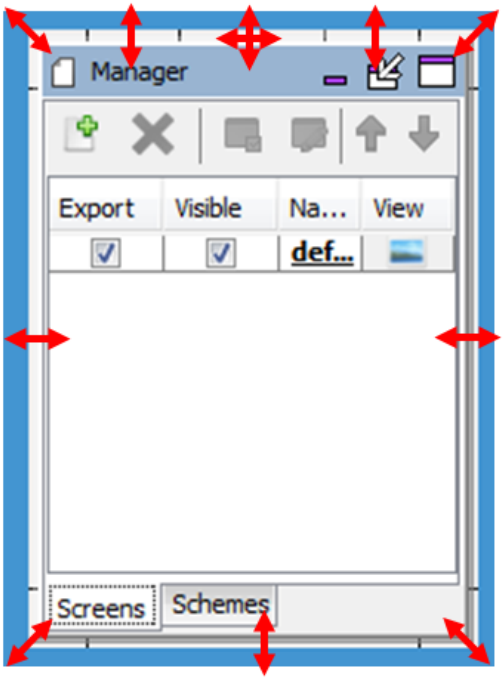
Note: In v2.03b of MPLAB Harmony, a third tab named Options, along with Screens and Schemes was available. These properties are now located within the *File* > *Settings* menu.

- Widget Tool Box – Available graphics widgets are shown on this panel. Widgets are added to the screen by selecting an icon and dragging or clicking. Widget properties are discussed in the Widget Properties section below.
- Properties Editor – All properties for the currently selected object are shown in this panel.
- The MHGC Output console is parked at the bottom of the Screen Designer window. This console panel can be used to debug problems when the Graphics Composer boots up or during its operation.

Each of the panels has a window tool icon at the upper right corner. Minimizing a panel parks it on the screen just like the Output Console. Undocking the panel creates a new, free floating window. Redocking returns a previously undocked window to its original location on the Screen Designer window.

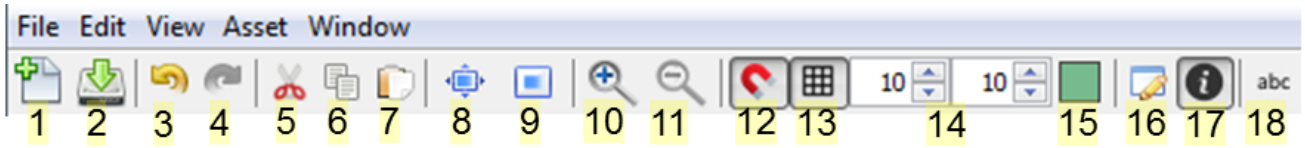


When a panel is undocked, its edges become active and support moving or manipulating the panel as an independent window.



Tool Bar

There are 18 tool bar icons on the Screen Designer Window, as described in the following figure.



Tool Icons:

1: Create New Design
2: Save Design
3: Undo
4: Redo
5: Cut Selected Object(s)
6: Copy Selected Object(s)
7: Paste Selected Object(s)
8: Canvas Size Dialog
9: Center View
10: Zoom In
11: Zoom Out
12: Toggle Line Snapping
13: Show Grid
14: X and Y Grid Size
15: Grid Color
16: Toggle Object Clipping
17: Toggle Screen Info
18: Select Text Preview Language

Create New Design brings up a New Project Wizard dialog that allows you to select anew the screen size, color mode, memory size, and project type. This will erase the currently displayed design.

Save Design saves the current graphics design.

Note: The target configuration's `configuration.xml` will not be updated to reflect these changes in the graphics design until one of the following events happens:

1. The application is regenerated in MHC,
2. The target configurations are changed in the MPLAB X IDE,
3. MPLAB X IDE is exited.

In items 2 and 3 you will be prompted to save the new configuration.

Undo and **Redo** manipulate changes in the screen design into internal MHC memory.

Cut/Copy/Paste support the manipulation of graphics objects (widgets).

Canvas Size Dialog brings up a dialog window allowing changes in the pixel width and height of the Screen Designer panel. (Note: Dimensions smaller than the display's dimensions are ignored).

Center View centers the panel's view of the screen.

Zoom In and Zoom Out allow you to change the scale of the Screen Designer's display of the current window. Currently this only supports coarse zooming (powers of two zooms in and out).

Toggle Line Snapping enables/disables line snapping when moving objects (widgets).

Show Grid turns the Screen Designer pixel grid on/off.

X and Y Grid Size adjust the pixel grid.

Grid Color selects the pixel grid color.

Toggle Object Clipping turns object clipping on/off.

Toggle Screen Info turns the display of screen information (X and Y axes) on/off.

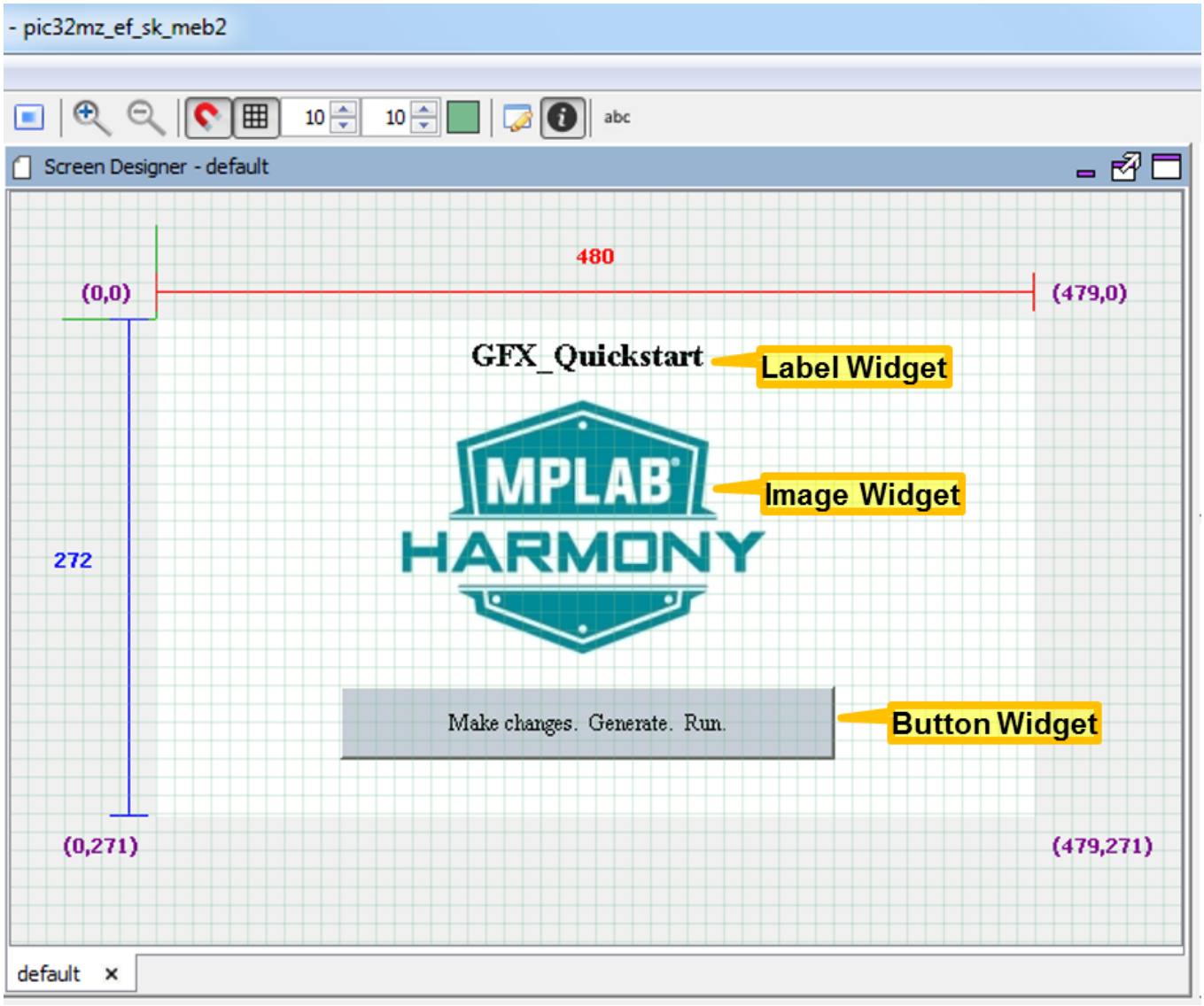
Select Text Preview Language changes the language used on all text strings shown, when the application supports more than one language.

Screen Designer Window

Most of the work of the MPLAB Harmony Graphics Composer is done using the Screen Designer. This section covers the basics of how a graphical user interface is designed using the screen designer.

Description

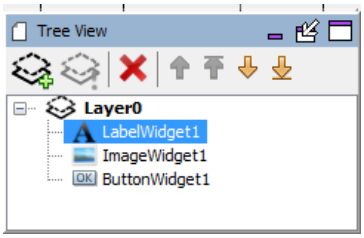
The following figure shows the Screen Designer window for the Aria Quickstart demonstration, with the pic32mz_ef_sk_meb2 configuration selected. (Load whatever configuration belongs to your board and follow along.)



The pixel dimensions of the display (480x272) are determined by the MHC Display Manager. Other configuration in Aria Quickstart can have different size displays (such as: 220x176, 320x24, or 800x480).

This demonstration has three widgets: a label containing the title string at the top, an image of the MPLAB Harmony logo in the middle, and a button containing the text string “Make changes. Generate. Run.” at the bottom. The label widget’s text string was first created using the String Assets window before it was assigned to the label widget. The image assigned to the image widget was first imported using the Image Assets. The string embedded in the button widget was also created using the String Assets window before it was assigned to the button widget.

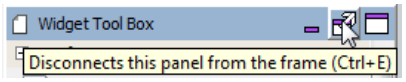
The Tree View panel organizes the display’s widgets into groups using layers. Every display has at least one layer and complex designs can have many more. Within the tree view, the order of layers and the order of widgets within a layer determine the draw order. Draw order goes from top to bottom. Top-most layers and widgets are drawn first and bottom-most are drawn last. Controlling draw order is one of the ways to improve graphics performance by minimizing redrawing.



Since the location of every widget within a layer is relative to the layer, you can move a layer’s worth of widgets by simply moving the layer. Layers also provide inheritance of certain properties from the layer to all the layer’s widgets.

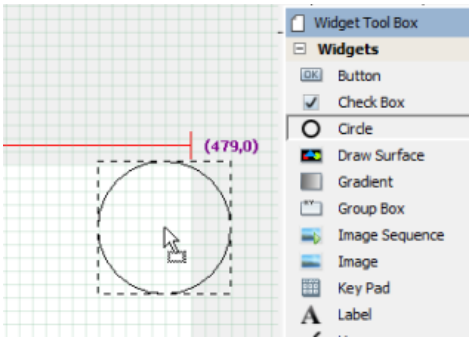
Exploring the Screen Designer Window

We can add another widget to this screen by launching the Widget Tool Box panel into a separate window.



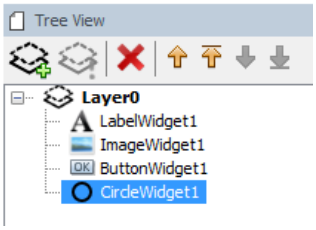
Next, drag a circle from the tool box onto the display. Find a place on the display for this new widget.

Besides dragging widgets onto the display, you can click on a widget in the Widget Tool Box, converting the cursor into that widget, and then click on the screen to drop the widget in place.

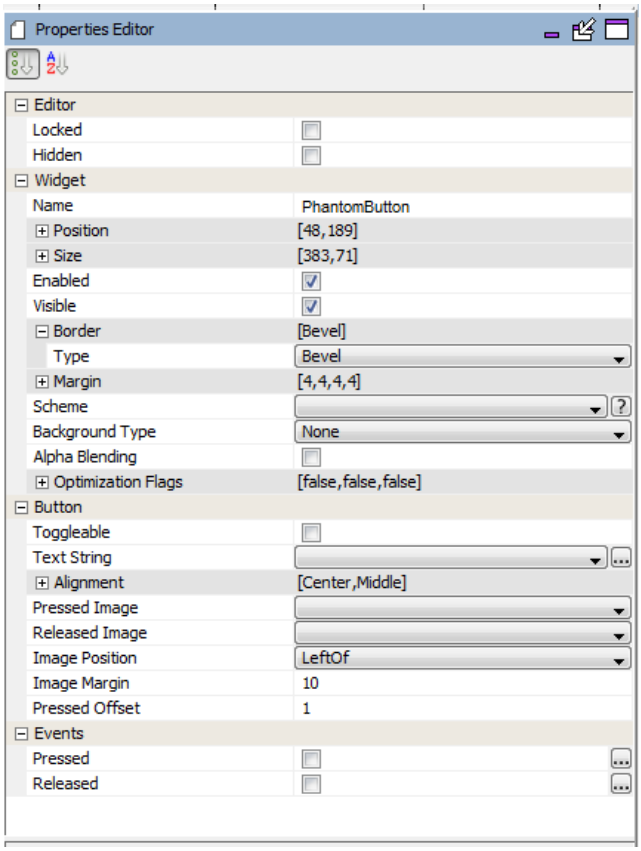


Your display should now look appear like the following figure.

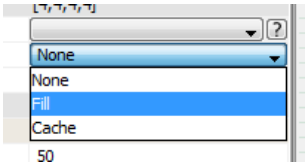
Note how the Tree View panel now shows the widget you just added.



Launch the Properties Editor for the circle.



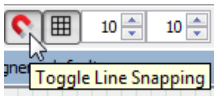
Next, change the fill property on the circle from “None” to “Fill”.

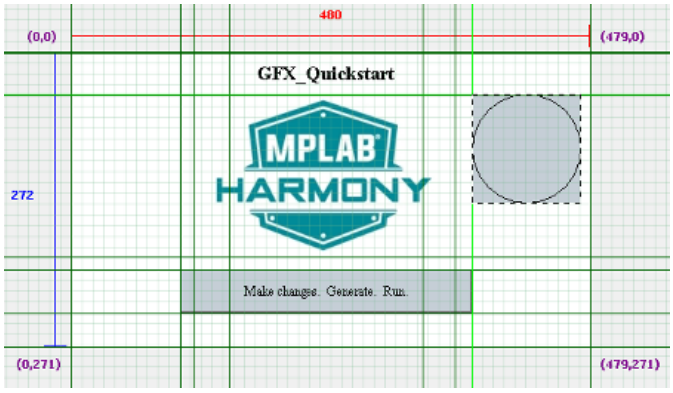


Note: If the properties in the Properties Editor shown are not for CircleWidget1, click on the circle widget to change the focus of the Properties Window.

When done, the screen should now appear, as follows.

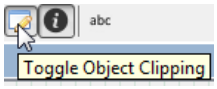
Turn on Line Snapping, which enables drawing guides to assist in aligning widgets on the display.

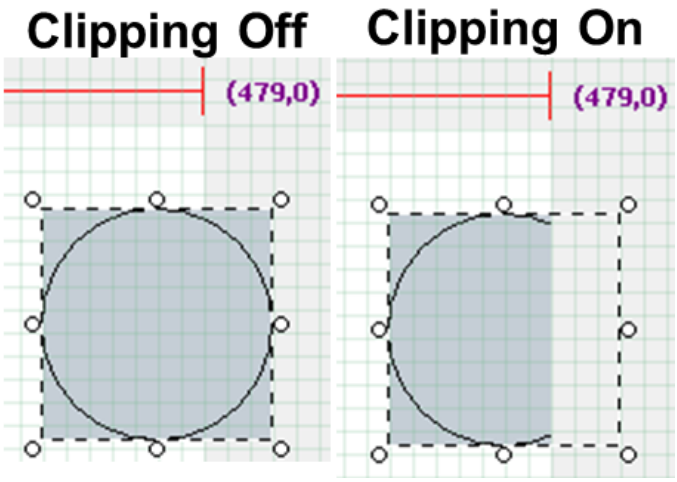




Next, turn on Object Clipping, which allows you to see how widgets are clipped by the boundaries of the layer that contains them.

Note: Clipping applies to layers, which can be smaller than the display.





To delete a widget, select the widget and press Delete on your keyboard or use the delete icon () on the Tree View panel.

For more hands-on exploration of graphics using the Aria Quickstart demonstration, see *Volume 1: Getting Started With MPLAB Harmony > Quick Start Guides > Graphics and Touch Quick Start Guides > Adding an Event to the Aria Quickstart Demonstration*.

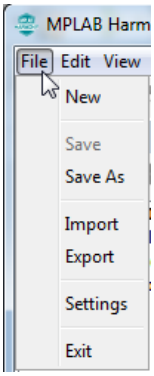
The steps to create a new MPLAB Harmony project with touch input on a PIC32MZ EF Starter Kit with the Multimedia Expansion Board (MEB) II display can be found in *Volume 1: Getting Started With MPLAB Harmony > Quick Start Guides > Graphics and Touch Quick Start Guides > Creating New Graphics Applications*.

Menus

This section provides information on the menus for the MPLAB Harmony Graphics Composer screen.

Description

File Menu



New – Same as the Create New Design tool icon.

Save – Same as the Save Design tool icon.

Save As – Supports exporting the design under a new name. By default, the name is `composer_export.xml`. See Importing and Exporting Graphics Data for more information.

Import - Reads in (imports) a previously exported design or a `./framework/src/system_config/{board_config}/configuration.xml` file that contains the graphics design to be imported. See Importing and Exporting Graphics Data for more information.

Export – Same as Save As. See Importing and Exporting Graphics Data for more information.

Settings – Brings up Project and User Settings dialog, including:

- Project Color Mode - How colors are managed
- Using a Global Palette
- Show Welcome Dialog
- Pre-emption Level – Allows for sharing of the device's cycles with other parts of the application
- Hardware Acceleration – Is graphics hardware accelerator enabled in software?

Exit – Closes the MHGC window and exits

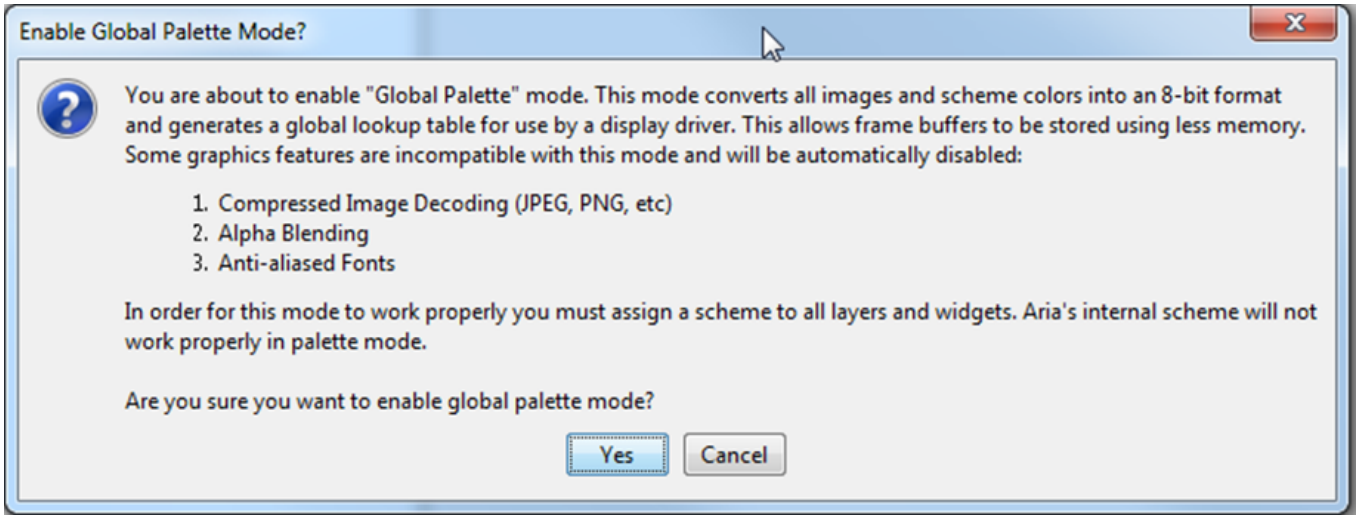
The choices for *Project and User Settings > Project Color Mode* are:

- GS_8 - 8-bit gray scale
- RGB_332 - Red/Green/Blue, 3 bits Red/Green, 2 bits Blue
- RGB_565 - Red/Green/Blue, 5 bits Red, 6 bits Green, 5 bits Blue
- RGBA_5551 - Red/Green/Blue/Alpha, 5 bits Red/ Green/Blue, 1 bit for Alpha Blending
- RGB_888 - Red/Green/Blue, 8 bits Red/Green/Blue
- RGBA_8888 - Red/Green/Blue/Alpha, 8 bits Red/Green/Blue/Alpha Blending
- ARGB_8888 - Alpha/Red/Green/Blue, 8 bits Alpha Blending/Red/Green/Blue

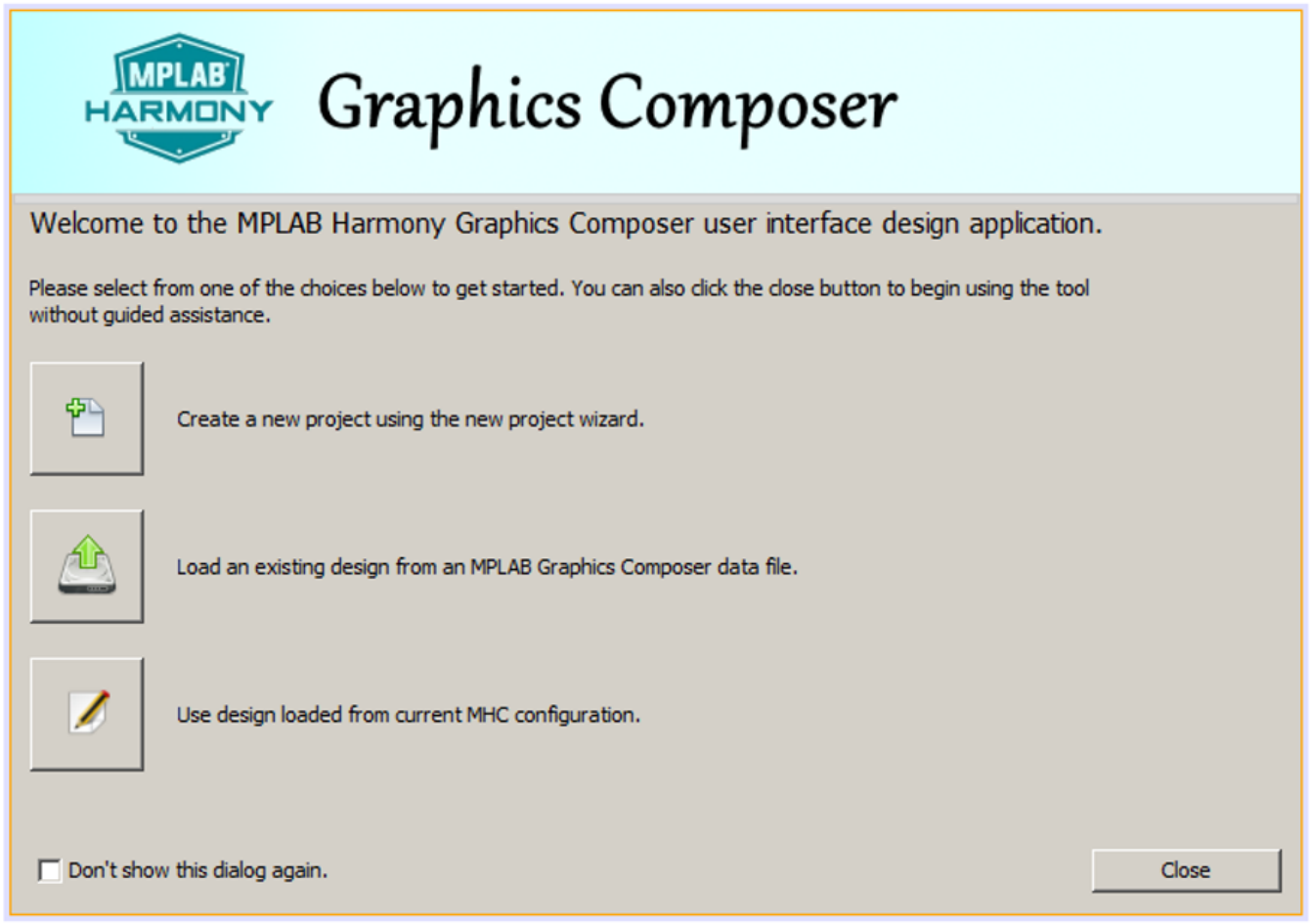
Ensure that the Project Color Mode chosen is compatible with the display hardware you are using; otherwise, the colors shown on the display will not match those shown on the Graphics Composer Screen Designer.

Using a Global Palette enables frame buffer compression for applications using the Low-Cost Controllerless (LCC) Graphics Controller or Graphics LCD (GLCD) Controller. If the global palette is enabled, you will have to change the MHC configuration of the Graphics Controller to match. For the LCC controller, enable "Palette Mode". For the GLCD controller, change the *Driver Settings > Frame Buffer Color Mode* to "LUT8".

If **Using a Global Palette** is enabled, the following warning appears.



If **Show Welcome Dialog** is enabled, the following welcome screen appears when launching MHGC.

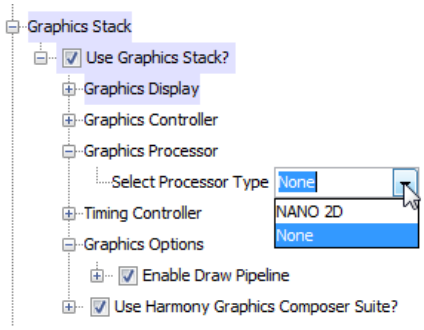


Note: If you are not creating a new project you can ignore this window.

When the **Preemption Level** is set to zero, all dirty graphics objects are refreshed before the graphics process relinquishes control of the device. (Dirty means needing a redraw.) With the level set to two, graphics provides maximum sharing with the rest of the application, at the cost of slower display refreshes. A level of one provides an intermediate level of sharing.

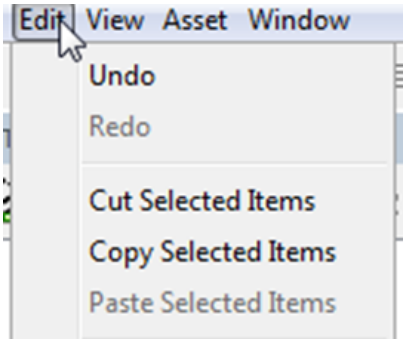
The **Hardware Acceleration** check box determines whether graphics uses the device's built-in graphics hardware accelerator in software.

Note: You must also specify the graphics hardware accelerator in the MPLAB Harmony Framework Configuration within the MHC Options tab. If the host device lacks a graphics processor, you will see a warning message when you try to select a processor that does not exist on your device.



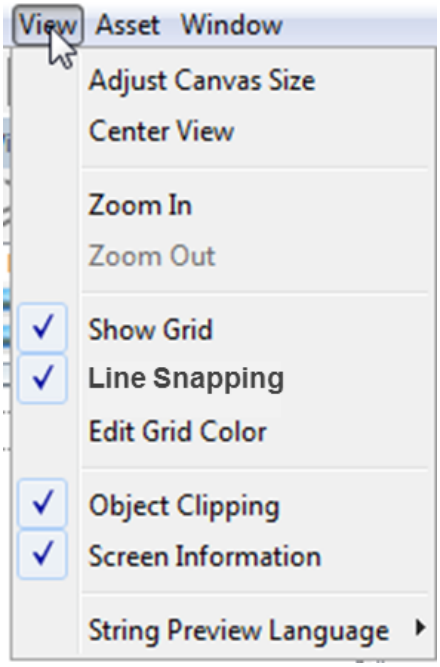
Edit Menu

This menu implements the same functions as the first seven tool icons.



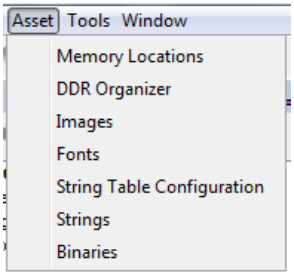
View Menu

This implements the same functions as the remaining tool icons.



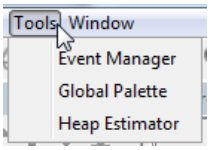
Asset Menu

These menu features are discussed in Graphics Composer Asset Management.



Tools Menu

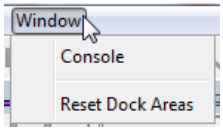
The Event Manager, Global Palette, and Heap Estimator are discussed in MHGC Tools.



Window Menu

Selecting **Console** opens the Output Console for the Graphics Composer. This console panel can be used to debug problems when the Graphics Composer boots up or during its operation.

Selecting **Reset Dock Areas** restores the MHGC panel configuration to the default setup by redocking all of the panels that have been undocked into separate windows.



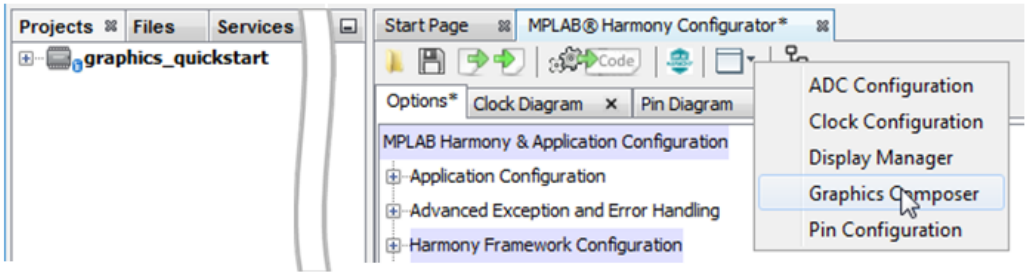
New Project Wizard

The New Project Wizard is launched from the Welcome dialog of the MPLAB Harmony Graphics Composer (MHGC), which supports the creation of a new graphics design, or the importing of an existing graphics design.

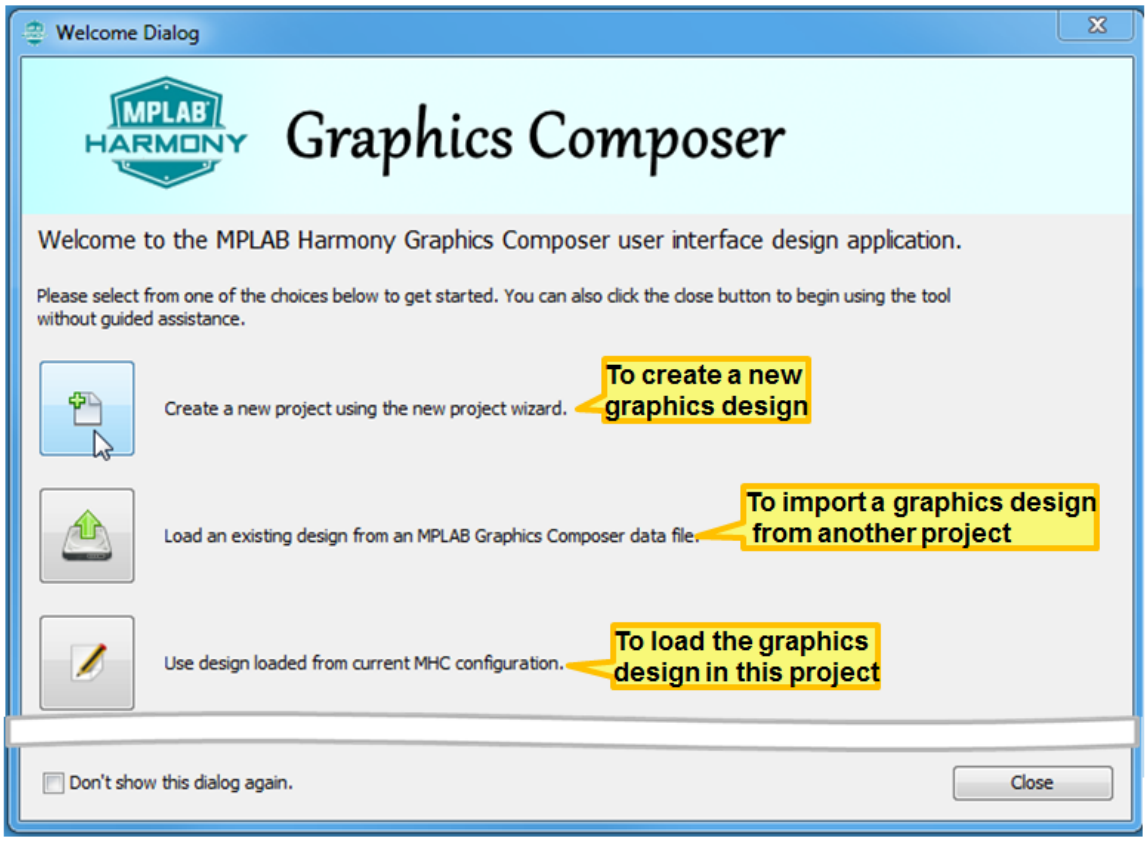
Description

Welcome Dialog window

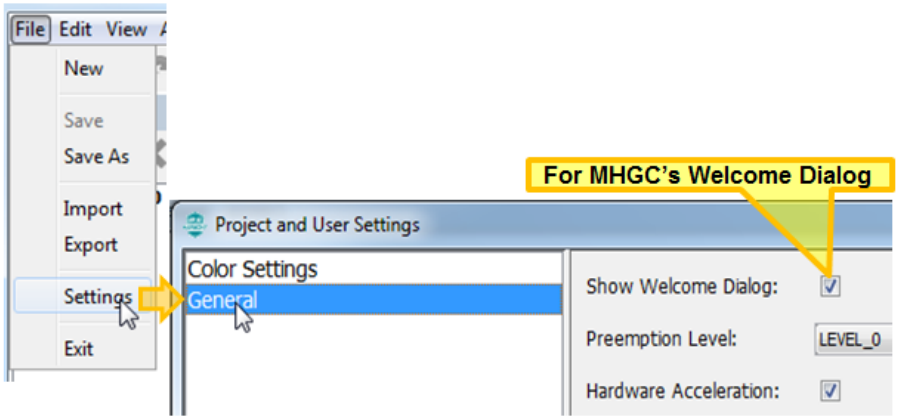
The Welcome dialog is launched when the Graphics Composer is chosen from the Launch Utility pull-down menu in the MPLAB Harmony Configurator (MHC).



The window has three options:



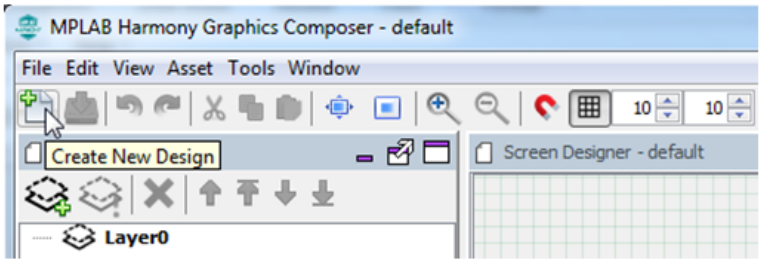
Note: If this window does not appear, it can be re-enabled from MHGC's *File* > *Settings* > *General* menu.



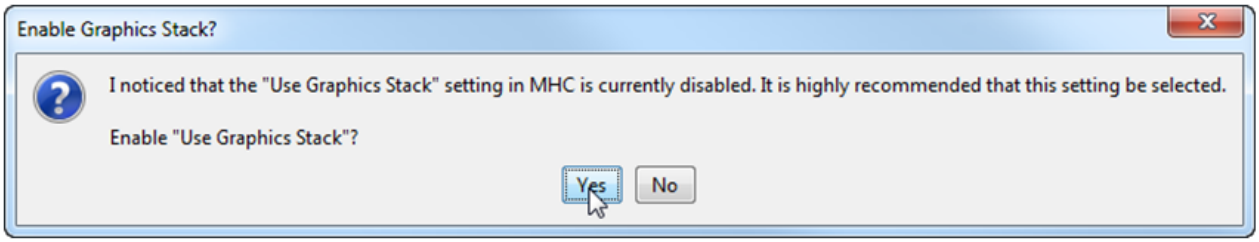
New Project Wizard Windows

Selecting the first icon in the Welcome dialog launches the New Project Wizard. There are four stages in the New Project Wizard: Color Mode, Memory Size, Project Type, and Finish.

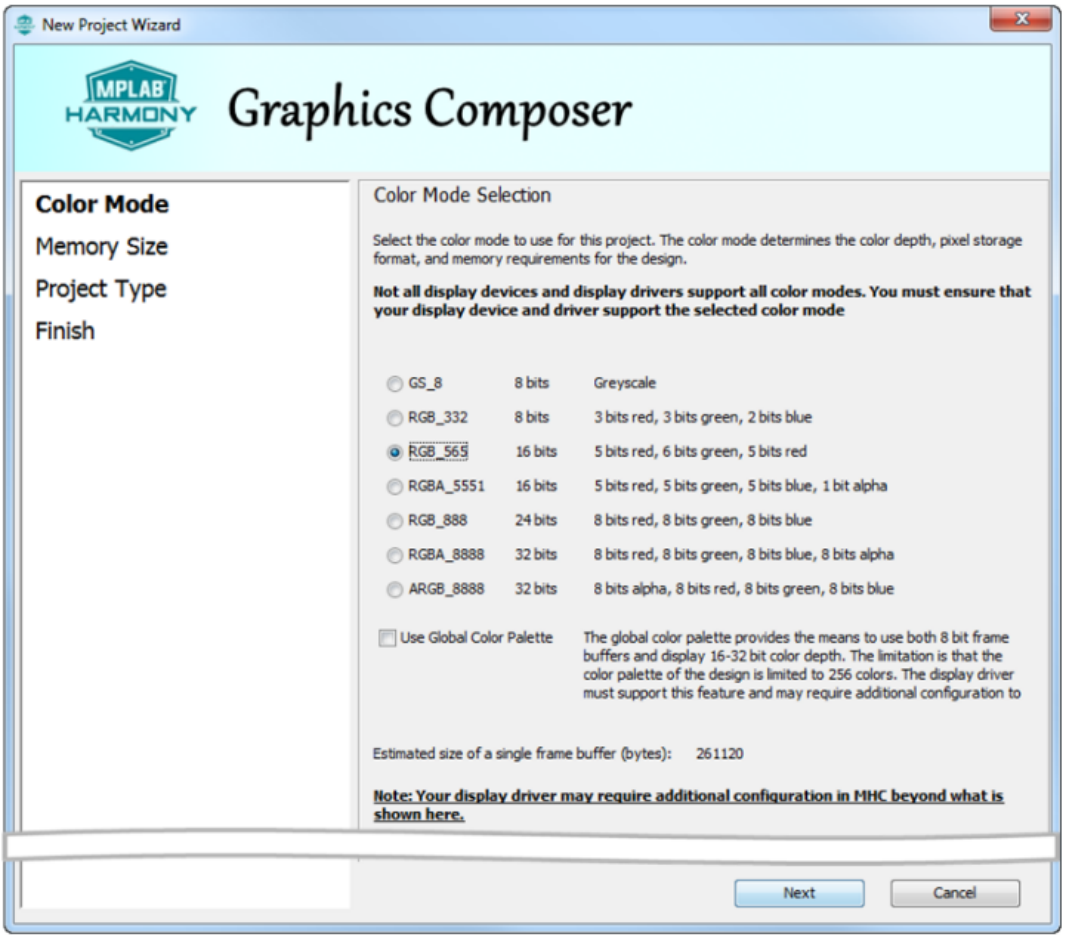
The New Project Wizard can also be launched from the first icon (Create New Design) of MHGC's tool bar:



If the Graphics Stack has not been enabled in MHC, an Enable Graphics Stack? dialog will appear to support enabling the Graphics Stack before proceeding:

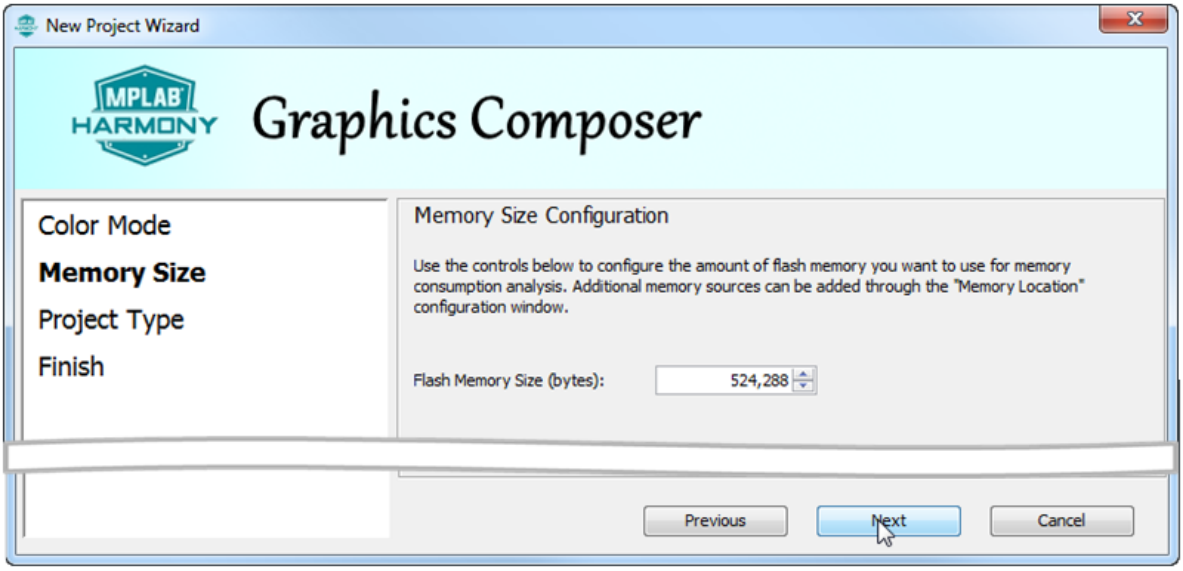


In the Color Mode stage you choose the Display Color Mode for the new graphics design:



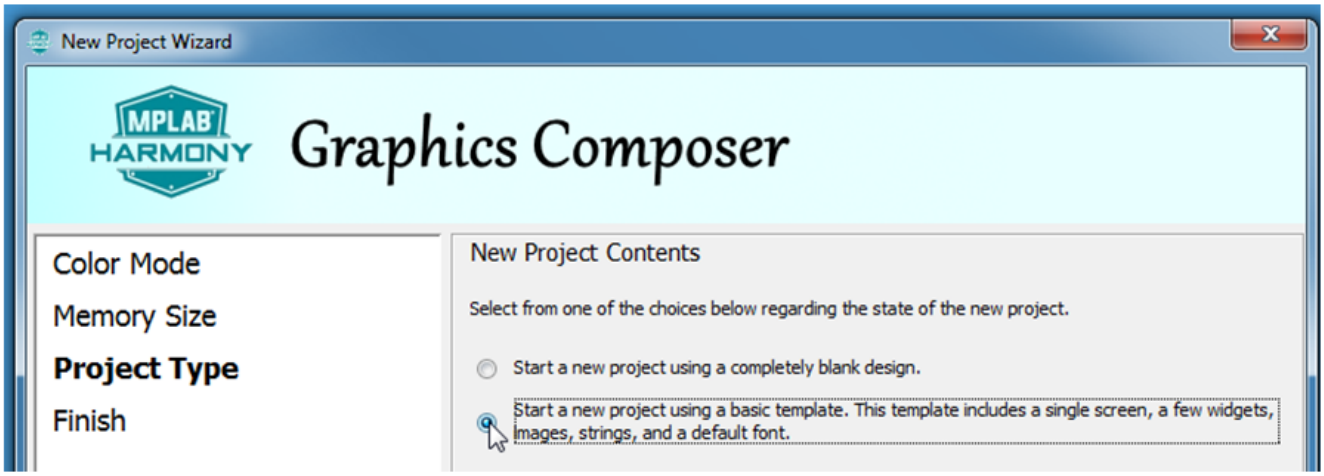
This choice must be supported by the graphics controller defined in the board support package of the project configuration. (If you make a mistake it can be corrected using MHGC's *File* > *Settings* > *Project Color Mode* menu.) Click **Next** moves the wizard on to the next stage.

The Memory Size stage configures the Program Flash allocated to memory use. This value is only used by the Graphics Composer's Asset menu Memory Configuration tool. The value used in the Memory Size stage can be updated using the Configuration sub-tab of the Memory Configuration tool window.

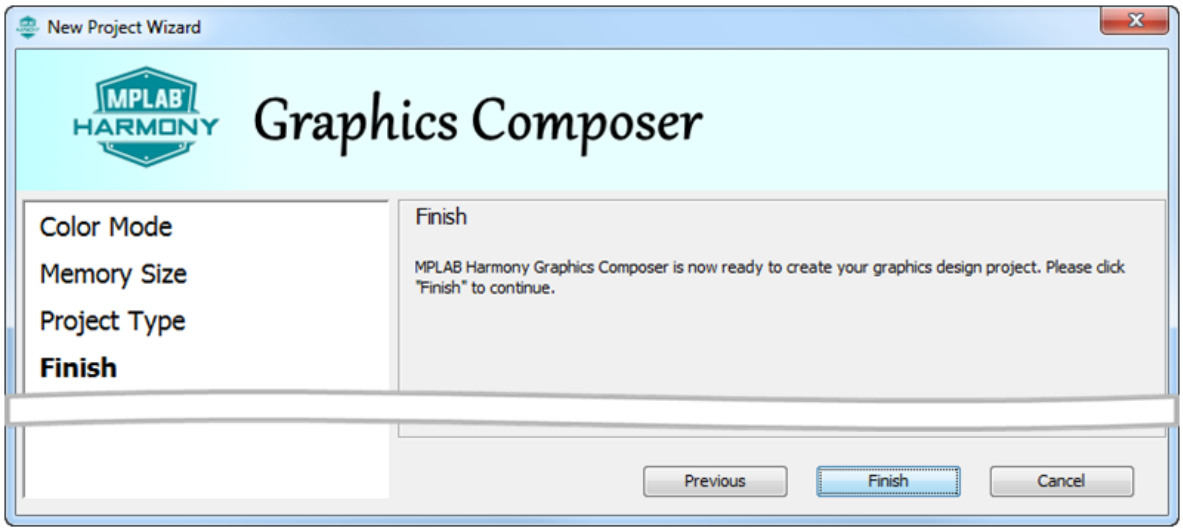


Clicking **Previous** returns to the Color Mode stage and clicking **Next** moves the wizard to the Project Type stage.

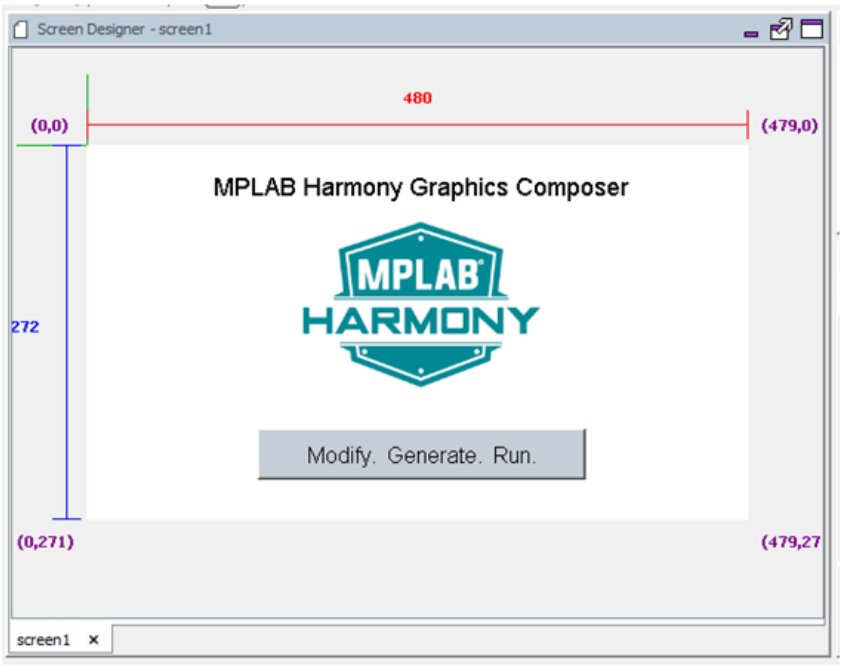
There are two choices at the Project Type stage: A completely blank design, and a template design with a few predefined widgets.



Clicking **Previous** returns to the Memory Size stage, and clicking **Next** moves the wizard to the Finish stage.



If the “Template” project type was chosen, MHGC’s Screen Designer will show:



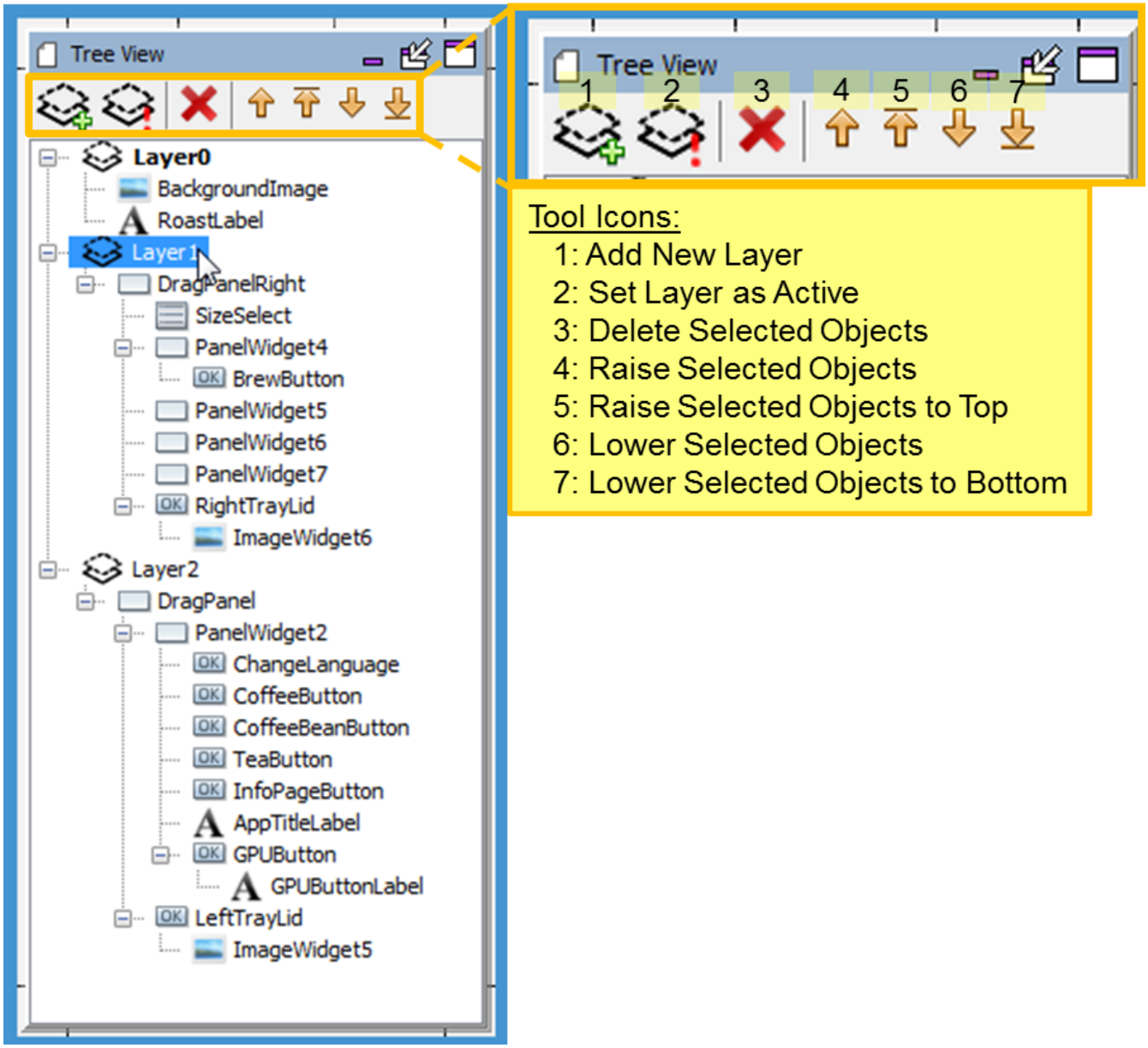
Tree View Panel

The organization of application widgets and layers, including draw order, is managed using this panel.

Description

Example Tree View

The following Tree View (from main screen of the Aria Coffee Maker demonstration shows the tree structure for a screen with three layers.



The tool icons for this panel support layers and managing screen objects (layers/widgets).

Drawing Order and Parent/Child Relationships

The Graphics Composer Tree View panel allows you to organize the widgets per screen in the desired drawing order (z-order). It also allows for the user to organize the widgets into parent – child hierarchies to allow for the paint algorithm to draw the groups together in event of motion or re-draw. Please note that this does not associate or group the widgets by functionality. (Example: a group of radio buttons might not belong to a common parent on the screen.) This parent-child relationship is limited to the widgets location on the screen, motion on the screen and the drawing order on the screen. (Exceptions to this general rule are the Editor > Hidden, Alpha Blending properties, and layer single versus double buffering. These apply to the parent and all the parent's children.)

The tree is traversed depth-first. This means that the z-order goes background (bottom of z-order) to foreground (top of z-order) as we go from top to bottom in the list of widgets, i.e., ImageWidget1, is the widget at the bottom of the z-order and the PanelWidget1 is the topmost widget on the z-order. The tree structure can be arranged and modified by dragging the widgets and releasing it under the desired parent/child. Also, the list can be modified by using the up/down arrows provided at the header of the Composer Widget tree window to traverse the tree.

Editor > Hidden Property for Layers

Setting *Editor* > *Hidden* hides the layer and all its children from the Graphics Composer Screen Designer but does not affect how the layer and its children are displayed when the application is running. This can be useful when designing complex screens with overlapping layers.

Alpha Blending Property for Layers

Enabling Alpha Blending allows you to control the transparency of a layer and all its children. You can experiment with Alpha Blending in the Aria Coffee Maker demonstration. Load the project, launch MHC, and then start the Graphics Composer Screen Designer. There are three layers (Layer0, Layer1, Layer2) in this demonstration. Layer1 (the drag panel on the right) and Layer2 (the drag panel on the left) have Alpha Blending enabled with Alpha Amount = 225. Setting the Alpha Amount to 255 is the same as disabling Alpha Blending (255 = no transparency). Setting the

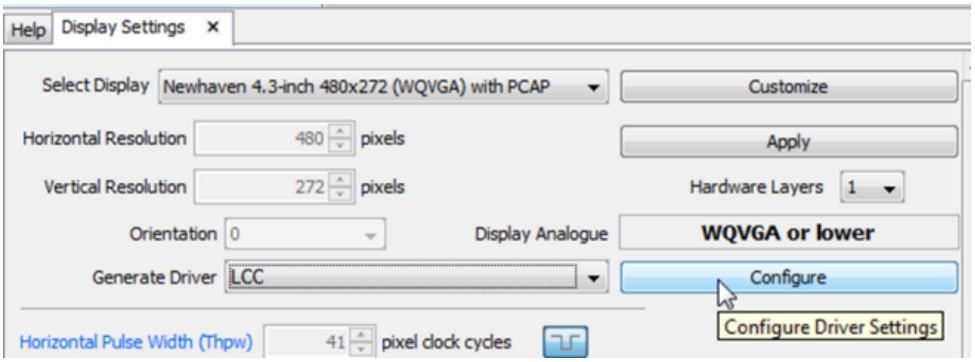
Alpha Amount to 0 makes the layer invisible (0 = full transparency, i.e., invisible).

The following figure shows the main screen with Alpha Blending = 225.

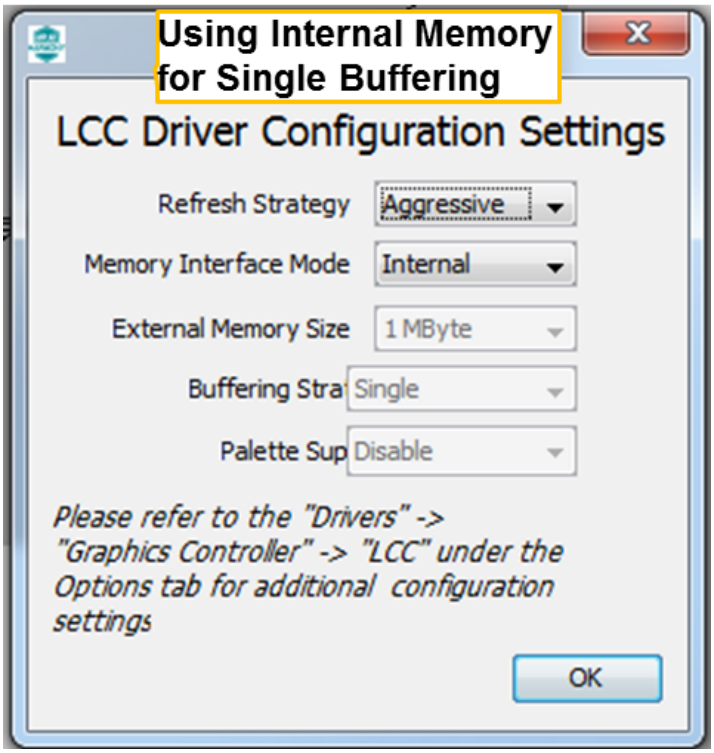
The following figure shows the main screen with Layer 2's Alpha Blending = 255.

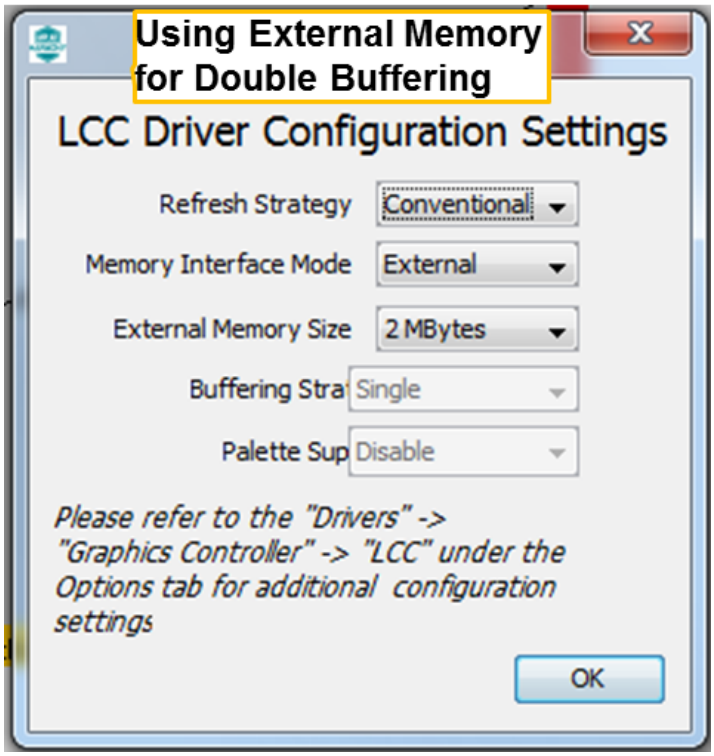
Double Buffering for Layers

Graphics double buffering for the LCC driver is enabled in the Display Manager's Display Setting screen when the application is changed to use external memory instead of internal. Click **Configure** to bring up the LCC Driver Configuration Settings Window.

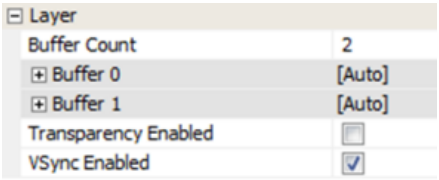


Configure the memory according to whether double buffering is to be enabled for the display's layer or layers.





Increasing the Buffer Count of a layer from 1 to 2 enables double buffering for the layer and all its child widgets. To prevent tearing on the display when switching from one buffer to the other, VSync Enabled should also be selected.

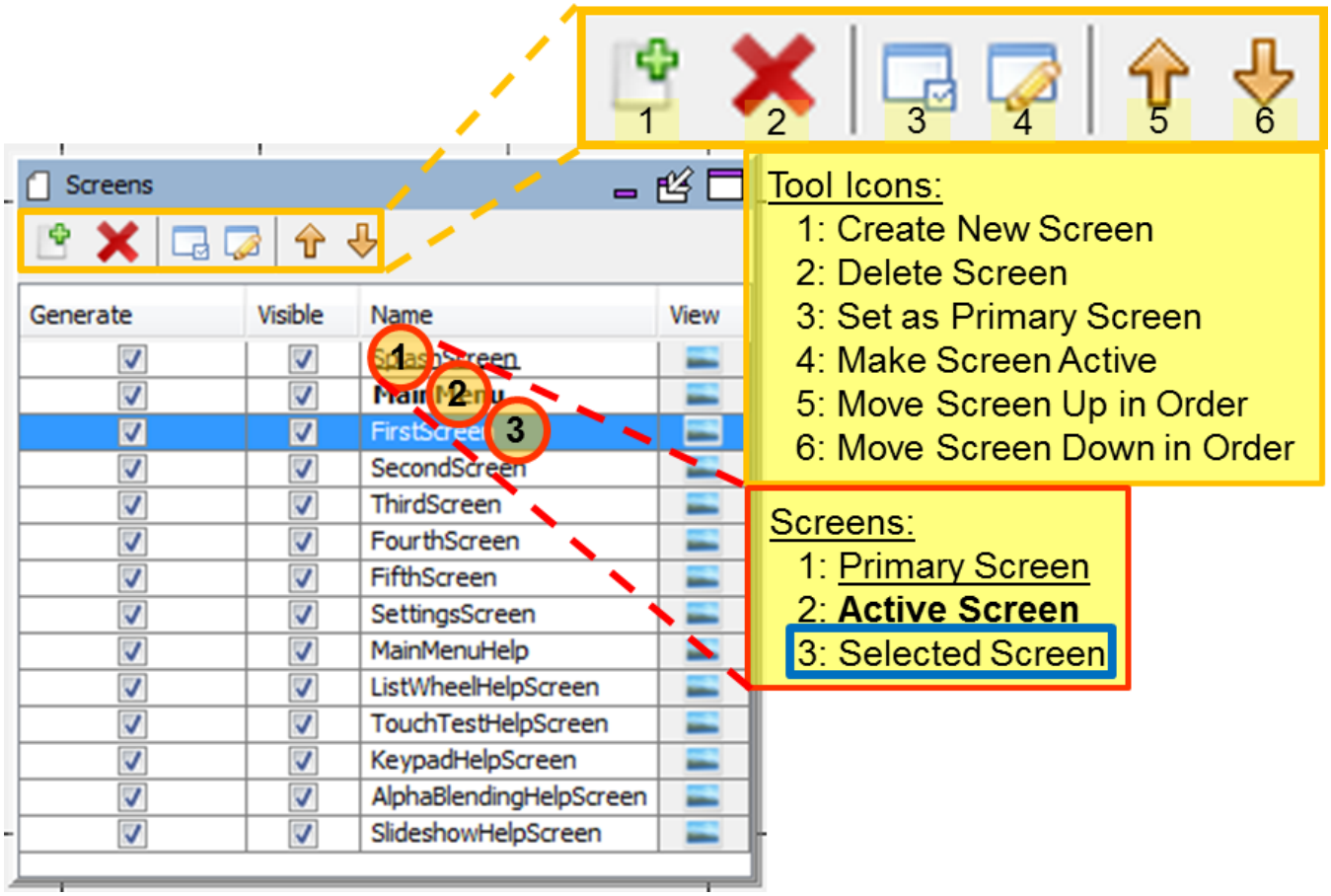


Screens Panel

Application screens are managed using the Screens Panel.

Description

The Screens panel tab manages all the application's screens, as shown in the following figure.



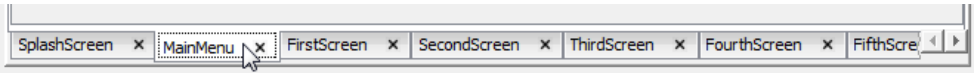
Note: These screens are examples from the Aria Showcase demonstration project

The underlined screen name identifies the primary screen (in this case, SplashScreen.) The **bold** screen name identifies the currently active screen in the Graphics Composer Screen Designer window (in this case MainMenu.) The blue background identifies the selected screen (i.e., the screen that is manipulated by the tool icons), in this case FirstScreen.

Window Toolbar

The window's tools icons support:

1. **Create New Screen** – Create a new screen. You will be prompted for the name of the new screen, which will appear at the bottom of the Screens list.
2. **Delete Screen** – Delete the selected screen. This removes the selected screen from the application.
3. **Set as Primary Screen** – Sets the selected screen as the default screen displayed by the application at boot-up.
4. **Make Screen Active** – This selected screen is displayed in the Screen Designer panel. You can also select the active screen by clicking on the screen's tab at the bottom of the Screen Designer panel.



5. **Move Screen Up in Order** – Moves the selected screen up in the list of screens, which is useful in organizing a large list of screens, but has no other significance.
6. **Move Screen Down in Order** – Moves the selected screen down in the list of screens.

Useful in organizing a large list of screens, but has no other significance.

Window Columns

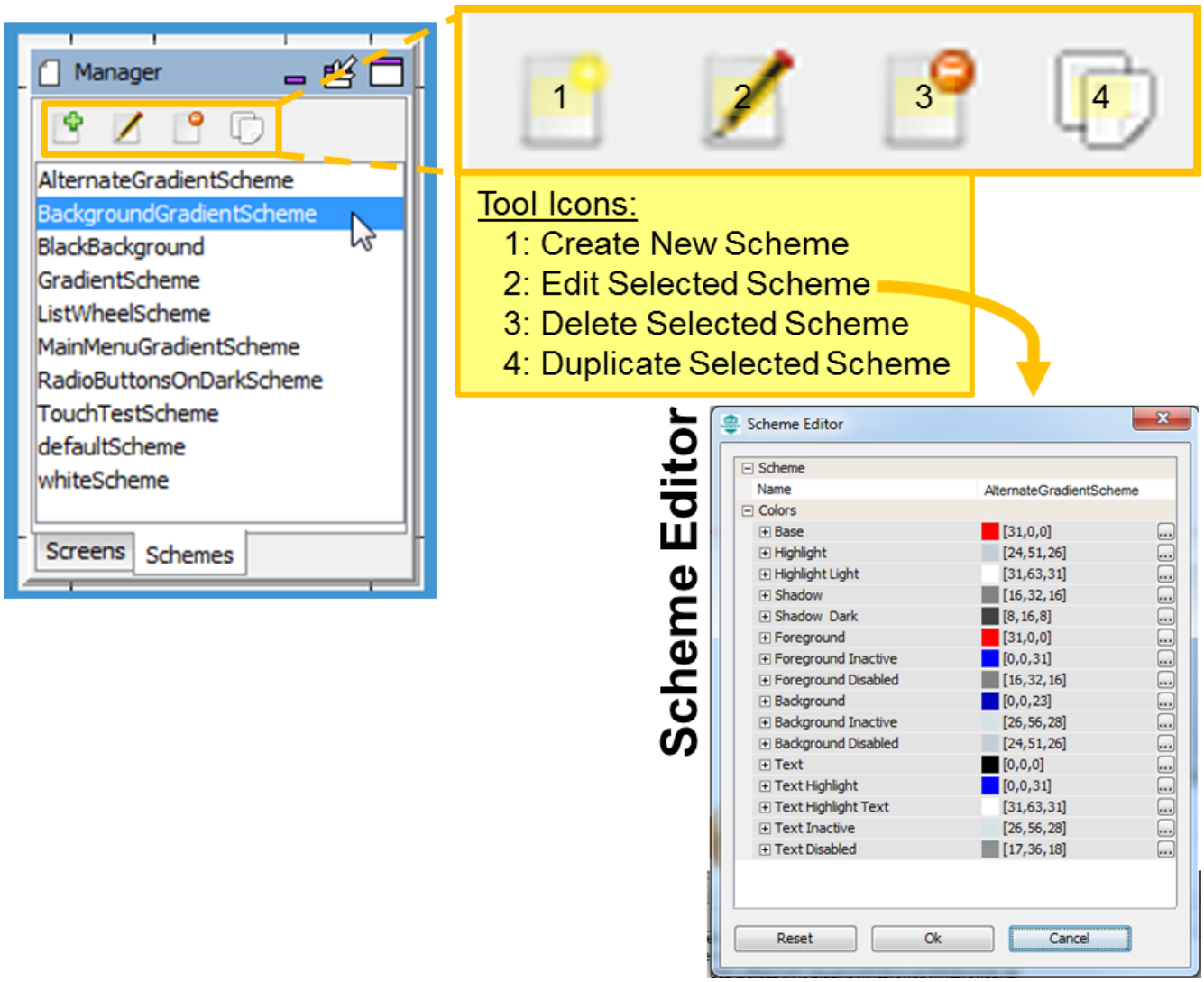
The **Generate** check box is used in selecting those screens that will be included in the application when MPLAB Harmony Configurator (MHC) generates/regenerates the application. (This, along with the Enabled check box for languages, allows customization of the application's build to support different end uses from the same project.) The **Visible** check box can be cleared to hide a screen from the sub-tabs located at the bottom of the Screen Designer. The **View** column provides a mouse-over preview of the screen.

Schemes Panel

Application color schemes are managed using the Schemes Panel.

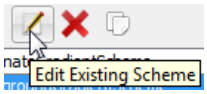
Description

Color schemes for the application's graphics are managed using the Schemes sub-tab.



Editing a Scheme

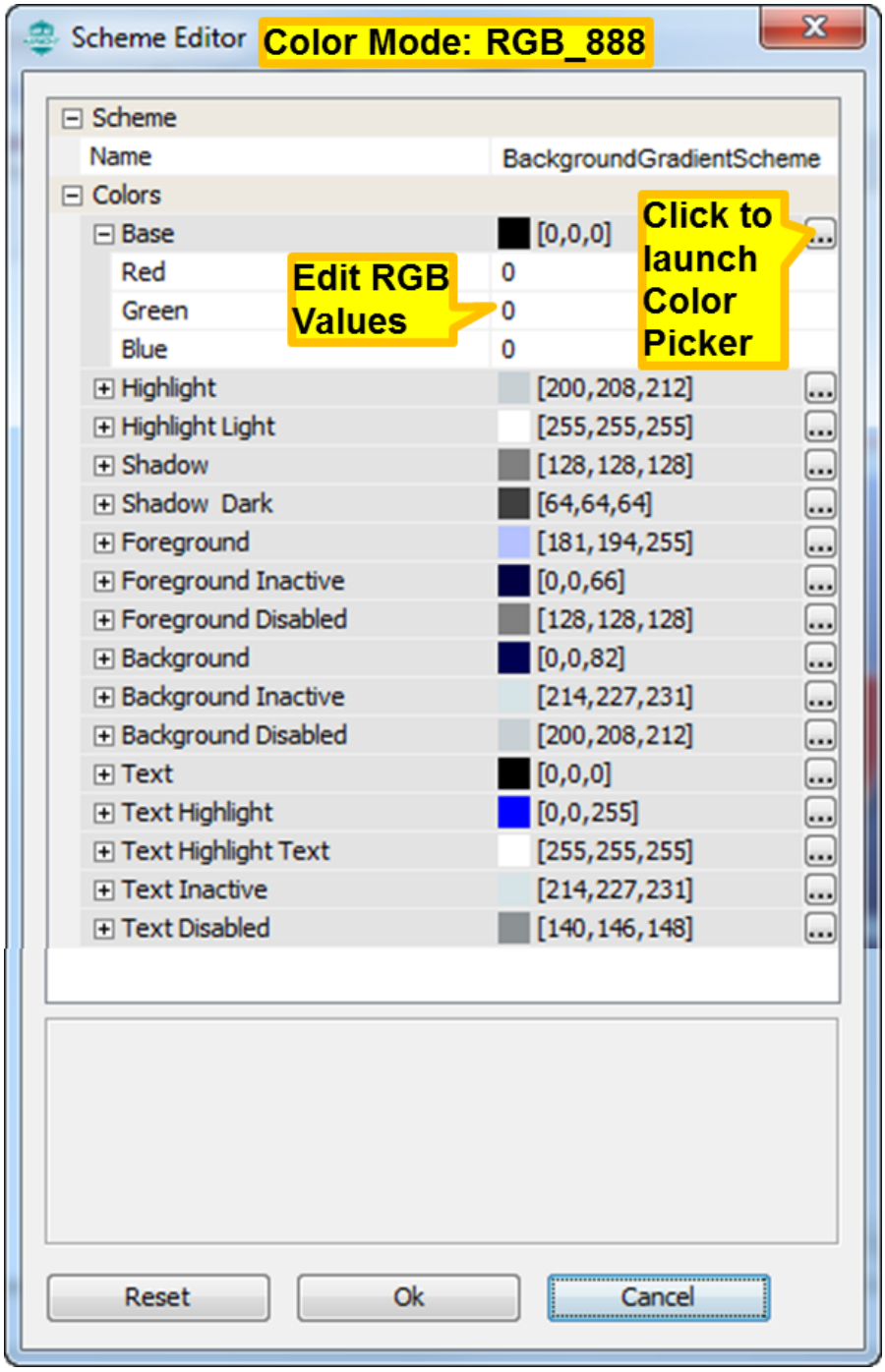
To edit an existing scheme, select the scheme from the list and click **Edit**.



The Scheme Editor dialog appears, which allows you to change the colors associated with this display scheme.

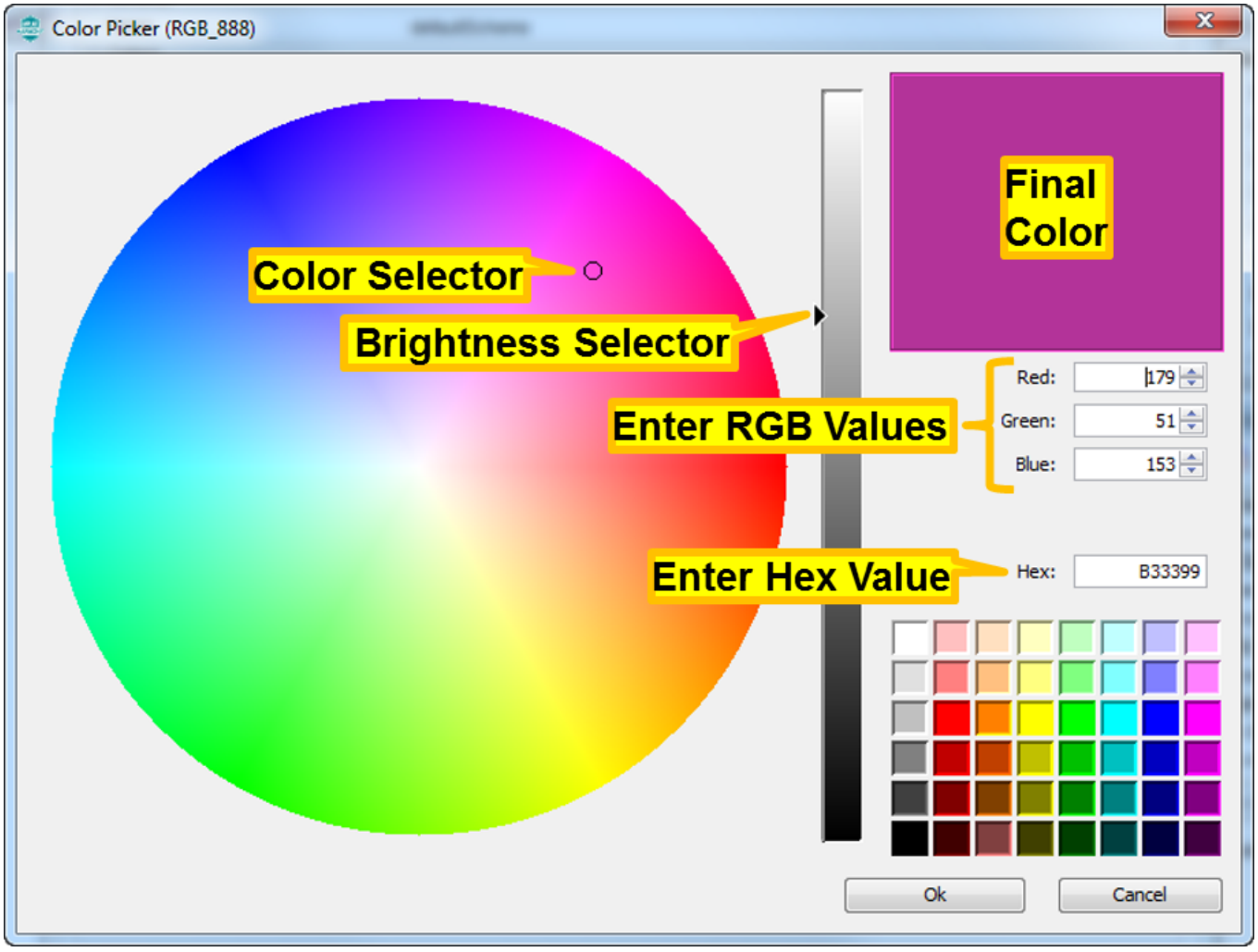
Scheme Editor

The Scheme Editor window supports editing the individual colors of a color scheme. Clicking the ellipsis (**...**) opens the Color Picker window.



Color Picker

The Color Picker window allows the user to easily select a color by providing a color wheel, brightness gauge, and some common predefined color choices. The user can change the individual color values or input a number in Hexadecimal format. The end result is displayed in the top right corner.



Options

Provides information on the defeatured Options window.

Description

In v2.03b, MPLAB Harmony Graphics Composer user interface provided a third window along with Screens and Schemes, named **Options**. Beginning with v2.04b of MPLAB Harmony, these options are now located within the *File* > *Settings* menu (see Menus for details).

Widget Tool Box Panel

The Widget Tool Box panel is the interface by which users add widgets into the screen representation.

Description

All the available graphics widgets are shown in the Widget Tool Box:

MPLAB Harmony Graphics Composer provides automatic code optimization by keeping track of the widgets that are currently being used. When MHC generates or regenerates the application, only the Graphics Library code necessary for your design is included in the project.

There are two primary methods for creating new widget objects: clicking and dragging. To add a new layer to a screen use the Screens sub-tab.

Click Method

The following actions can be performed by using the Click method:

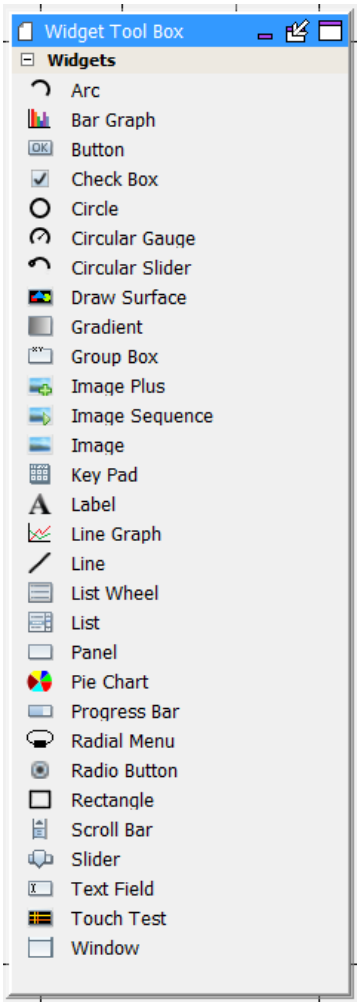
- Clicking an item selects it as active. Users can then move the cursor into the screen window and view a representation of the object about to be added.
- Left-clicking confirms the placement of the new object
- Right-clicking aborts object creation
- Clicking the active item again deactivates it

Drag Method

Dragging and dropping a tool item into the Screen Designer Window creates a new instance of an object. When dragging a tool item, releasing the cursor outside of the Screen Designer Window cancels the drag operation.

Widget List

The Graphics Composer Tool Box is the interface by which users add widgets into the screen representation.



Widget	Example Application
Arc	aria_showcase_reloaded
Bar Graph	aria_showcase_reloaded
Button	aria_adventure and many others, including aria_quickstart
Check Box	aria_showcase_reloaded, aria_video_player
Circle	None
Circular Gauge	aria_showcase_reloaded, aria_oven_controller
Circular Slider	aria_showcase_reloaded
Draw Surface	None
Gradient	aria_showcase (background)
Group Box	aria_video_player
Image	aria_quickstart
Image Plus	aria_oven_controller
Image Sequence	aria_showcase, aria_basic_motion
Key Pad	aria_showcase, aria_touchadc_calibrate

Label	aria_quickstart
Line	aria_video_player, ./aps/examples/3rd_party_display
Line Graph	aria_showcase_reloaded
List Wheel	aria_showcase
List	aria_video_player
Panel	aria_video_player
Pie Chart	aria_showcase_reloaded
Progress Bar	aria_flash
Radial Menu	aria_radial_menu, aria_showcase_reloaded
Radio Button	aria_showcase
Rectangle	aria_benchmark
Scroll Bar	None
Slider	aria_video_player
Text Field	aria_showcase
Touch Test	aria_showcase, aria_touchadc_calibrate, ./apps/examples/3rd_party_display
Window	None

Click Method

The following actions can be performed by using the Click method:

- Clicking an item selects it as active. Users can then move the cursor into the screen window and view a representation of the object about to be added.
- Left-clicking confirms the placement of the new object
- Right-clicking aborts object creation
- Clicking the active item again deactivates it.

Drag Method

Dragging and dropping a tool item into the Screen Designer Window creates a new instance of an object. When dragging a tool item, releasing the cursor outside of the Screen Designer Window cancels the drag operation.

Automatic Code Optimization

MPLAB Harmony Graphics Composer keeps track of the types of widgets that are used and updates the MHC Tree constantly to ensure that only the Graphics Library code necessary for your design is included in the project.

Widgets

Widgets can be configured by using the Properties Editor on the right side of the MHGC interface. Each widget has multiple properties to manage their appearance as well as their functioning. Most properties related to appearance are common between widgets, though some widgets require specific property entries.

Arc – A graphical object in the shape of an arc. The arc thickness can be set and filled.

Bar Graph – A graphing widget that shows data in categories using rectangular bars.

Button - A binary On and Off control with events generation for Press and Release state.

Check Box - A selection box with Checked and Unchecked states, and associated events.

Circle - A graphical object in the shape of a circle.

Circular Gauge – A circular widget that operates like a gauge, where the hand/needle position indicates a value.

Circular Slider – A circular widget that can change values based on external input like touch. The slider is filled based on the value of the widget relative to the maximum value.

Draw Surface - A container with a callback from its paint loop. a draw surface lets the application have a chance to make draw calls directly to the HAL during LibAria's paint loop.

Gradient - A draw window that can be associated with a gradient color scheme. This allows for color variation on the window.

Group Box - A container with a border and a text title. With respect to functionality, a group box is similar to a window.

Image Sequence - A special widget that allows image display on screen to be scheduled and sequenced. Select the images to be displayed, and the order for display. A timer to trigger the transitions must be created by calling the image sequence APIs to show the next image from the timer callback function.

Image - Allows an image to be displayed on screen. The size and shape of the widget decides the visible part of the image, as scaling is not enabled for images at this time.

Image Plus - Allows an image to be displayed on screen. The image can be resized (aspect ratio lock is optional). The widget can be set to accept two-finger touch input.

Key Pad - A key entry widget that can can be designed for the number of entries divided as specified number of rows and column entries. The widget has a key click event that can be customized.

Label - A text display widget. This does not have any input at runtime capability. A Text Field widget serves that purpose.

Line - A graphical object in the shape of a line.

Line Graph – A graphing widget that shows data in categories using points and lines.

List Wheel - Allows multiple radial selections that were usually touch-based selections and browsing.

List - Allows making lists of text and image items. The list contents, number of items, and the sequence can be managed through a List Configuration dialog box in the Properties box.

Panel - A container widget that is a simpler alternative to DrawSurface as it does not have the DrawSurface callback feature.

Pie Chart – A graphing widget that shows data entries as sectors in a circle.

Progress Bar - Displays the progress pointer for an event being monitored through the "Value Changed" event in the Properties Editor.

Radial Menu - A widget that groups any number of images into an elliptical carousel. It can configured as a touch interactive image carousel or interface menu.

Radio Button - A set of button widgets that are selected out of the group one at a time. The group is specified by the Group property in the Properties Editor.

Note: The radio buttons in the same group must have the same group number specified in their properties.

Rectangle - A graphical object in the shape of a rectangle.

Scroll Bar - Intended to be used with another relevant widget such as the List Wheel to scroll up and down. It has a callback each time the value is changed. The callback allows users to trigger actions to be handled on the scroll value change event.

Slider - Can change values with an external input such as touch. Event callbacks on value change are also available through the Properties Editor.

Text Field - Text input can be accepted into the text field from an external input or from a widget such as keypad. Event 'Text Changed' in the Properties Editor is used for accepting the input.

Touch Test - Allows tracking of touch inputs. Each new touch input is added to the list of displayed touch coordinates. The input is accepted through the 'Point Added' event callback in the Properties Editor.

Window - A container widget similar to the Panel but has the customizable title bar.

Properties Editor Panel

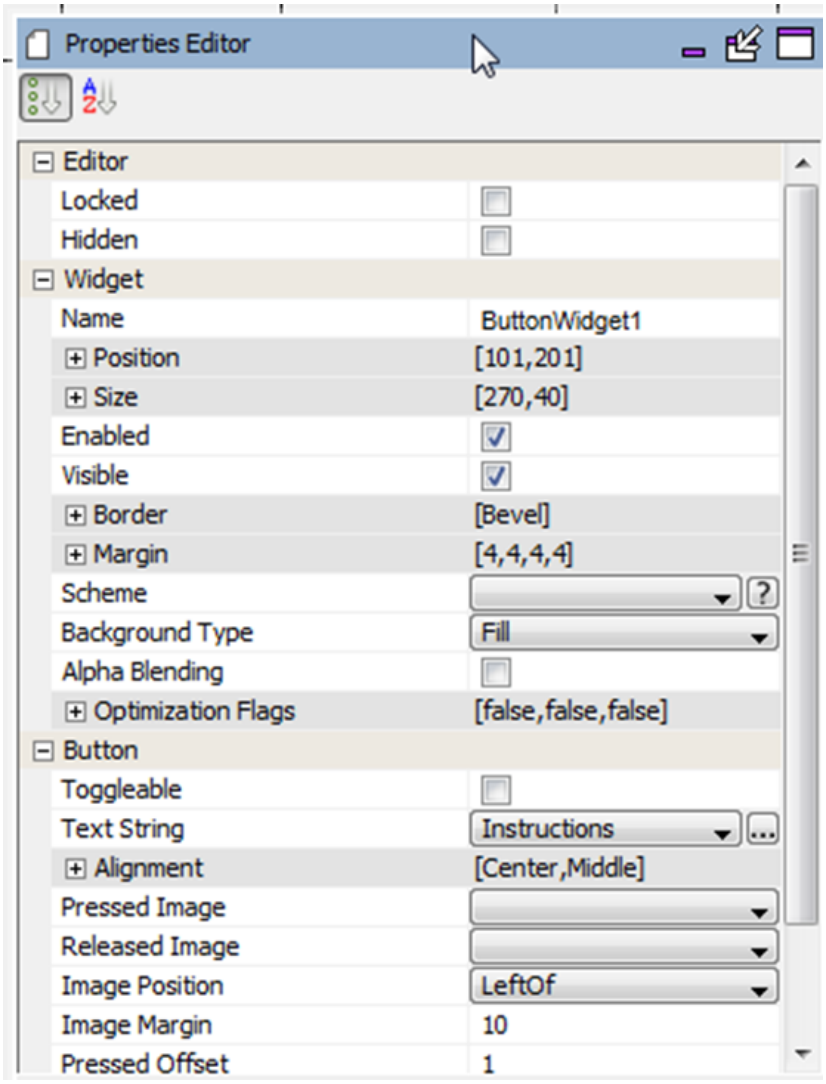
The properties for all layers and widgets are managed using this panel.

Description

The Properties Editor displays options for the currently-selected object (layer or widget), or the options for the active screen if no objects are selected. To edit an option: left-click the value in the right column and then change the value. Some values have an ellipsis that will provide additional options. In the previous case, the ellipsis button will display the Color Picker dialog.

Some properties, like the screen width and height, are locked and cannot be edited. Other properties offer check boxes and combo-type drop-down box choices. Some properties are grouped together like the Position and Size entries. Individual values of the group can be edited by expanding the group using the plus symbol. For example, the following figure shows properties for a Button Widget.

A new support feature is the ? icon to the right of the Scheme pull-down, which brings up an “Scheme Helper” for the widget showing how it is colored when using a Bevel border. For a more complete description of widget coloring, see Widget Colors.



Object Properties

Provides information on widget, layer, and screen properties.

Description

Object Properties and Event Actions

Each widget has a structured tree of properties, visible under the MPLAB Harmony Configurator window on the right of the standard window setup within MPLAB X IDE. Most widget properties have a Related Event action that can be use in an event or macro to change or set a property from the application.

Each widget has 3-4 property sets:

Editor – Controls the behavior of layers and widgets under the MPLAB Harmony Graphics Composer Suite Editor.

Property Name	Type	Description	Related Event Actions
Locked	Boolean	Locks the object (widget), preventing changes by the designer. Only affects the object (widget) in the editor.	N/A
Hidden	Boolean	Hides the widget and its children in the designer window. Only affects the appearance of the widget in the editor.	N/A
Active	Boolean	For layers only. Sets the layer as active. Any objects (widgets) added to the screen will be added to this layer.	N/A
Locked to Screen Size	Boolean	For layers only. Locks the layer size to the size of the display's screen.	N/A

Widget – Controls the behavior of screens, layers, and widgets on the display.

Property Name	Type	Description	Related Event Actions
Name	String	Editable name for each object. By default, widgets are named NameWidget1, …,NameWidgetN. For example: ButtonWidget1, ButtonWidget2, … .	N/A
Position	[X,Y] Pair of Integers	Location on the layer of the upper left corner of the widget or the location on the display of the upper left corner of the layer. Measured in display pixels. X is measured from left-to-right and Y is measured from up-to-down from the upper left corner of the parent object (typically a Layer or Panel).	Adjust Position, Set X Position, Set Y Position
Size	[X,Y] Pair of Integers	X: Width, Y: Height of object, in display pixels.	Adjust Size, Set Size, Set Width, Set Height
Enabled	Boolean	Is the object enabled? Disabled objects are not built into the display's firmware.	Set Enabled
Visible	Boolean	Is the object visible by default? Object visibility can be manipulated in firmware using laWidget_GetVisible and laWidget_SetVisible.	Set Visible
Border	Widget Border	Choices are: { None | Line | Bevel }.	Set Border Type
Margin	Integer	Four integers ([Left,Top,Right,Bottom]) defining the widget's margins on the display, in display pixels.	Set Margins
Scheme	-	Color scheme assigned to the layer or widget. Blank implies the default color scheme.	Set Scheme
Background Type	-	Sets the background of the layer or widget. Choices are { None | Fill | Cache }. In MPLAB Harmony v2.03, this type was Boolean. Now, Off = None, On = Fill. With Fill selected, the widget's background is one solid color. With Cache selected, a copy (cache) of the framebuffer is created before the widget is drawn and this cache is used to fill the background of the widget. This supports transparent widgets in front of complex widgets, such as JPEG images. Instead of rerendering the JPEG image, it is just drawn from the cache.	Set Draw Background
Alpha Blending	Boolean	Is alpha blending enabled for this layer or widget and all of its children? If enabled, specify the amount of alpha blending as an 8-bit integer. Zero makes the object invisible, whereas 255 makes the background invisible.	N/A

Widget Advanced – Advanced control of layers and widgets

Optimization Sub-Property Name	Type	Description	Related Event Actions
Draw Once	Boolean	Indicates that the widget should draw once per screen Show Event. All other attempts to invalidate or paint the widget will be rejected.	N/A
Force Opaque	Boolean	Provides a hint to the renderer that the entire area for this widget is opaque. Useful for widgets that may use something like an opaque image to fill the entire widget rectangle despite having fill mode set to None. This can help reduce unnecessary drawing.	N/A
Local Redraw	Boolean	Provides a "hint" to the widget's renderer that the widget is responsible for removing old pixel data. This can avoid unnecessary redrawing.	N/A

Important! Use Local Redraw only if you know what you're doing!

Widget Name (e.g., Button Check Box, Circle, etc.) – Optional properties tied to each widget. See **Dedicated Widget Properties and Event Actions**.

Events – Associates widget events with event call-backs. For example, you can enable and specify a button pressed event and button release event for the Button widget.

For each event you specify:

- Enabled/Disabled Check box – To enable or disable (default) the event.

- Event Callback – Selected from the Event Editor Action List.

There are additional Event actions that do not correspond to any specific property:

- Set Parent – Set the parent of the object, including no parent.

Dedicated Widget Properties and Event Actions

Arc Widget

Property Name	Type	Description	Related Event Actions
Radius	Integer	The outside radius of the arc.	Set Radius
Start Angle	Integer	The starting angle of the arc in degrees.	Set Start Angle
Center Angle	Integer	The center angle of the arc in degrees. A positive angle draws the arc counter-clockwise from the start angle. A negative angle draws clockwise.	Set Center Angle
Thickness	Integer	The thickness of the arc fill, measured from the radius to center. (radius – thickness) determines the inside radius.	Set Thickness
Round Edge	Boolean	Draws round arc edge.	Set Round Edge

Bar Graph Widget

Property Name	Type	Description	Related Event Actions
Stacked	Boolean	Stacks the bars for the entries in a category	Set Stacked Bars
Tick Length	Integer	The length, in pixels, of the ticks on each axis	Set Tick Length
Fill Graph Area	Boolean	Fills the graph area with scheme base color	Fill Graph Area
Value Axis Configuration • Maximum Value • Minimum Value • Tick Interval • Subtick Interval • Show Ticks • Tick Position • Show Tick Labels • Show Subticks • Subtick Position • Show Gridlines • String Set	 Integer Integer Integer Integer Boolean Enum Boolean Boolean Enum Boolean String Asset	Configures the value (Y) axis The maximum value of the axis The minimum value of the axis The intervals between major ticks The interval between minor ticks Show/Hide the major ticks Position of major ticks on the value axis. Choices are: {Inside | Center | Outside} Show/Hide the tick labels Show/Hide the minor ticks Position of minor ticks on the value axis. Choices are: {Inside | Center | Outside} Show/Hide the gridlines The string asset containing the numeric characters for the tick labels. The asset must contain the characters for numbers 0 to 9.	Set Max Value Set Min Value Set Tick Interval Set Subtick Interval Show Value Axis Ticks Set Value Axis Ticks Position Show Value Axis Labels Show Value Axis Subticks Set Value Axis Subticks Position Show Value Axis Gridlines Set Labels String
Category Axis Configuration • Show Tick • Show Category Labels • Tick Position	 Boolean Boolean Enum	Configures the category (X) axis Show/Hide the ticks Show/Hide the category labels Position of the ticks on the category axis. Choices are: {Inside | Center | Outside}	Show Category Axis Ticks Show Category Axis Labels Set Category Axis Ticks Position
Category Configuration Dialog	(See Description)	The Category Configuration Dialog lets users add categories to the line graph. The following properties can be set: • Label – String Asset. The label to show for each category	None
Data Configuration Dialog	(See Description)	The Data Configuration Dialog lets users add and configure data series to the line graph. The following properties can be set: • Scheme – Scheme. The color scheme of the data series • Category Values – Integer. Values in series for each category	None

Button

Property Name	Type	Description	Related Event Actions
Toggleable	Boolean	Is button toggle enabled?	Set Toggleable

Pressed	Boolean	If Toggleable is enabled, provide default state of the button. This can be used to see the colors of an asserted button.	Set Press State
Text String	-	Select widget's text string from the Select String Dialog.	Set Text
Alignment: • Horizontal • Vertical	-	Text string alignment within the button object. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment
Pressed Image	-	Select image used for pressed state. Default: no image.	Set Pressed Image
Released Image	-	Select image used for pressed state. Default: no image.	Set Released Image
Image Position	-	Position of image relative to button text. Choices are: { LeftOf | Above | RightOf | Below | Bottom }.	Set Image Position
Pressed Offset	Integer	Offset of button contents when pressed. In Pixels. The X and Y position of the button contents is offset by this amount.	Set Pressed Offset

Check Box

Property Name	Type	Description	Related Event Actions
Text String	-	Select widget's text string from the Select String Dialog.	Set Text
Alignment: • Horizontal • Vertical	-	Text string alignment within the button object. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment
Checked	Boolean	Default state of the check box.	Set Check State
Unchecked Image	-	Select image used for widget's unchecked state. Default: no image.	Set Unchecked Image
Checked Image	-	Select image used for the widget's checked state. Default: no image.	Set Checked Image
Image Position	-	Position of image relative to check box text. Choices are: : { LeftOf | Above | RightOf | Below | Bottom }.	Set Image Position
Image Margin	Integer	Space between image and text. In Pixels.	Set Image Margin

Circle

Property Name	Type	Description	Related Event Actions
X	Integer	X offset of circle's center, from widget's upper left hand corner, in pixels.	N/A
Y	Integer	Y offset of circle's center, from widget's upper left hand corner, in pixels.	N/A
Radius	Integer	Circle's radius, in pixels.	Set Radius

Circular Gauge Widget

Property Name	Type	Description	Related Event Actions
Radius	Integer	The outside radius of circular gauge.	Set Radius
Start Angle	Integer	The starting angle of the circular gauge in degrees.	Set Start Angle
Center Angle	Integer	The canter angle of the circular gauge in degrees. A positive value draws the gauge counter-clockwise. Clockwise if negative.	Set Center Angle
Start Value	Integer	The start value of the circular gauge.	Set Start Value
End Value	Integer	The end value of the circular gauge.	Set End Value
Value	Integer	The value of the circular gauge.	Set Value
String Set	String Asset	The string asset containing the numeric characters for the tick labels. The asset must contain the characters for numbers 0 to 9.	-

Major Ticks Configuration • Ticks Visible • Tick Length • Tick Value • Tick Labels Visible	 Boolean Integer Integer Boolean	Configures the major ticks. Shows/Hides the major ticks. The length of ticks in pixels. The interval between ticks. Shows/Hides the major tick labels.	 Show/Hide Ticks Set Tick Length Set Tick Value Show/Hide Tick Labels
Hand Configuration • Hand Visible • Hand Radius • Center Circle Visible • Center Circle Radius • Center Circle Thickness	 Boolean Integer Integer Integer Integer	Configures the gauge hand/needle. Shows/Hides the gauge hand/needle. Sets the length of the hand in pixels Shows/Hides the hand center circle. Sets the radius of the center circle in pixels Sets the thickness of the center circle in pixels.	 Show/Hide Hand Set Hand Radius/Length Show/Hide Center Circle Set Center Circle Radius Set Center Circle Thickness
Advanced Configuration	-	Additional widget configuration options for adding minor ticks, labels and arcs.	-
Minor Ticks Configuration Dialog	(See Description)	The Minor Ticks configuration lets users add minor ticks to the widget. The following properties can be set: • Start Value – Integer. The value where the first tick starts • End Value – Integer. The value where the last tick ends • Interval – Integer. The interval between ticks • Radius – The radius in pixels where the ticks will be drawn from • Length – The length of the ticks in pixels, drawn from the radius towards the center • Scheme – The color scheme for the ticks	None
Minor Tick Labels Configuration Dialog	(See Description)	The Minor Ticks configuration lets users add minor tick labels to the widget. The following properties can be set: • Start Value – Integer. The value where the first tick label is drawn • End Value – Integer. The value where the last tick ends • Interval – Integer. The interval between ticks • Radius – Integer. The radius, in pixels, where the tick labels will be drawn from • Position – Enum, choices are {Outside | Inside}. Position of the label relative to the radius • Scheme – The color scheme for the ticks	None
Arcs Configuration Dialog	(See Description)	The Arcs configuration lets users draw arcs in the gauge widget. The arcs can be used to colorize regions or range of values in the gauge. The following properties can be set for each arc: • Type – Enum, choices are {VALUE | ANGLE}. A value type arc is drawn relative to the values in the gauge. An angle type arc is draw based on the angles and is not affected by the values in the gauge. • Start – Integer. The start value or angle of the arc • End – Integer. The start value or angle of the arc • Thickness – Integer. The thickness of the arc in pixels, filled inward from the radius towards the center • Radius – Integer. The radius of the arc in pixels • Scheme. The color scheme of the arc	None

Circular Slider Widget

Property Name	Type	Description	Related Event Actions
Radius	Integer	The outside radius of circular slider.	Set Radius
Start Angle	Integer	The start angle of the circular slider, in degrees.	Set Start Angle
Start value	Integer	The start value of the circular slider.	Set Start Value
End Value	Integer	The end value of the circular slider.	Set End Value
Value	Integer	The value of the circular slider.	Set Value

Border Circle Configuration • Show Outside Circle • Outside Circle Thickness • Show Inside Circle • Inner Circle Thickness	 Boolean Integer Boolean Integer	Configures the border circle. Shows/Hides the outside circle border. The thickness of the outside circle border in pixels. Shows/Hides the inside circle border. The thickness of the inside circle border in pixels.	Show/Hide Outside Border Set Outside Border Thickness Show/Hide Inside Border Set Inside Border Thickness
Active Area Configuration • Fill Active Slider Area • Round Edges • Active Slider Area Thickness • Inner Circle Thickness	 Boolean Boolean Integer Integer	Configures the slider active area. Fills the active slider area. Draws a round edge for the active area. The thickness of the slider active area in pixels. The thickness of the inside circle border in pixels.	Show/Hide Active Arc Area Set Round Edges Set Active Arc Area Thickness Show/Hide Inactive Arc Area
Button Configuration • Show Circular Button • Sticky Button • Touch on Button Only • Circular Button Radius • Circular Button Thickness	 Boolean Boolean Boolean Integer Integer	Configures the slider button. Shows/Hides the circular slider button. If set, the button sticks when it reaches the start/end values. If set, the widget responds to touches within the button area only. The radius of the circular button in pixels. The thickness of the of the circular button border in pixels.	Show/Hide Circular Button Set Sticky Button None Set Circular Button Radius Set Circular Button Thickness

Draw Surface – No additional properties.

Gradient

Property Name	Type	Description	Related Event Actions
Direction	-	Gradient draw direction. Choices are: { Right | Down | Left | Up }.	Set Direction

Group Box

Property Name	Type	Description	Related Event Actions
Text String	-	Select widget's text string from the Select String Dialog.	Set Text
Alignment	-	Text string alignment within the widget. Choices are: { Left|Center|Right }.	Set Alignment

Image Sequence

Property Name	Type	Description	Related Event Actions
Sequence Configuration Dialog	-	Specify image sequence by using the Image Sequence Configuration Dialog window.	Set Entry Image, Set Entry Horizontal Alignment, Set Entry Vertical Alignment, Set Entry Duration, Set Image Count
Starting Image	Integer	Selects the first image to be shown.	Set Active Image
Play By Default	Boolean	Will image sequence play automatically?	N/A
Repeat	Boolean	Should the image sequence repeat?	Set Repeat Additional related event actions: , Show Next, Start Playing, Stop Playing.

Image Widget

Property Name	Type	Description	Related Event Actions
Image	-	Select image used.	Set Image
Alignment: • Horizontal • Vertical	-	Image alignment within the image object. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment

Image Plus Widget

Property Name	Type	Description	Related Event Actions
Image	-	Select Image used	Set Image
Resize To Fit	Boolean	Resize the image to fill the size of the widget area	Toggles option to best fit the image to the widget area
Interactive	Boolean	Makes the widget interactive, allowing the image to be translated, stretched and zoomed	Toggles option to permit two-finger gestures to interact with the widget

Key Pad

Property Name	Type	Description	Related Event Actions
Row Count	Integer	Number of key pad rows.	None.
Column Count	Integer	Number of key pad columns.	None.
Key Pad Configuration Dialog	(see Description)	The Key Pad dialog window has the following: • Width – Integer. Width of each key, in pixels. • Height – Integer. Height of each key, in pixels. • Rows – Integer. Number of key rows. A duplicate of Row Count. • Columns – Integer. Number of key columns. A duplicate of Column Count.	None. None. None. None.
-	-	Selecting one of the keys on the key pad diagram displays the Cell Properties for that key: • Enabled – Boolean. Disabled cells (keys) are made invisible. • Text String – Select key's text string from the Select String Dialog. • Pressed Image – Select image used for pressed state. Default: no image. • Released Image – Select image used for released state. Default: no image. • Image Position – Position of image relative to key text. Choices are: { LeftOf | Above | RightOf | Below | Behind }. • Image Margin – Integer. Space between image and text. In Pixels. • Draw Background – Boolean. Controls whether the key should fill its background rectangle. • Editor Action – Select the generic editor action that fires when the key is clicked. Choices are: { None | Accept | Append | • Editor Value String Other Key Event Actions:	Set Key Enabled Set Key Text Set Key Pressed Image Set Key Released Image Set Key Image position Set Key Image Margin None. Set Key Action Set Key Value Set Key Background Type

Label

Property Name	Type	Description	Related Event Actions
Text String	-	Select widget's text string from the Select String Dialog.	Set Text
Alignment: • Horizontal • Vertical	-	Text string alignment within the widget. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment

Line

Property Name	Type	Description	Related Event Actions
Start X	Integer	X start of line, in pixels, from upper left hand corner of the widget.	Set Start Point Position
Start Y	Integer	Y start of line, in pixels, from upper left hand corner of the widget.	Set Start Point Position
End X	Integer	X end of line, in pixels, from upper left hand corner of the widget.	Set End Point Position.
End Y	Integer	Y end of line, in pixels, from upper left hand corner of the widget.	Set End Point Position.

Line Graph Widget

Property Name	Type	Description	Related Event Actions
Stacked	Boolean	Stacks the values of the entries in a category	Set Stacked Points
Tick Length	Integer	The length of the ticks on each axis	Set Tick Length
Fill Graph Area	Boolean	Fills the graph area with scheme base color	Fill Graph Area
Fill Series Area	Boolean	Fills the series area with series scheme base color	Fill Series Area
Value Axis Configuration • Maximum Value • Minimum Value • Tick Interval • Subtick Interval • Show Ticks • Tick Position • Show Tick Labels • Show Subticks • Subtick Position • Show Gridlines • String Set	 Integer Integer Integer Integer Boolean Enum Boolean Boolean Enum Boolean String Asset	 Configures the value (Y) axis The maximum value of the axis. The minimum value of the axis. The intervals between major ticks. The interval between minor ticks. Show/Hide the major ticks. Position of major ticks on the value axis. Choices are: {Inside | Center | Outside}. Show/Hide the tick labels. Show/Hide the minor ticks. Position of minor ticks on the value axis. Choices are: {Inside | Center | Outside}. Show/Hide the gridlines. The string asset containing the numeric characters for the tick labels. The asset must contain the characters for numbers 0 to 9.	Set Max Value Set Min Value Set Tick Interval Set Subtick Interval Show Value Axis Ticks Set Value Axis Ticks Position Show Value Axis Labels Show Value Axis Subticks Set Value Axis Subticks Position Show Value Axis Gridlines Set Labels String
Category Axis Configuration • Show Tick • Show Category Labels • Tick Position	 Boolean Boolean Enum	 Configures the category (X) axis Show/Hide the ticks Show/Hide the category labels Position of the ticks on the category axis. Choices are: {Inside | Center | Outside}	Show Category Axis Ticks Show Category Axis Labels Set Category Axis Ticks Position
Category Configuration Dialog	(See Description)	The Category Configuration Dialog lets users add categories to the line graph. The following properties can be set: • Label – String Asset. The label to show for each category	None
Data Configuration Dialog	(See Description)	The Data Configuration Dialog lets users add and configure data series to the line graph. The following properties can be set: • Scheme – Scheme. The color scheme of the data series • Point Type – Enum. The point indicator to use for the series. Choices are: {None | Circle | Square} • Fill Points – Boolean. Fills the points with series scheme foreground color • Draw Lines – Boolean. Draws lines between points in the series using series scheme foreground color • Category Values – Integer. Values in series for each category	None

List

Property Name	Type	Description	Related Event Actions
Selection Mode	-	Select list selection mode. Choices are: {Single|Multiple|Contiguous}.	Set Selection Mode
Allow Empty Selection	Boolean	Is a list selection allowed to be empty?	Set Allow Empty Selection
Alignment	-	Horizontal text alignment. Choices are: { Left | Center | Right }.	Set Item Alignment
Icon Position	-	Position of list icons relative to list text. Choices are: { LeftOf | RightOf }.	Set Icon Position
Icon Margin	-	Space between icon and text, in pixels.	Set Icon Margin

List Configuration Dialog	-	Defines the string and icon image for each entry in the list.	Set Item Icon, Set Item Icon (actually sets item text). Additional Related Event Actions: Deselect All Items, Insert Item, Remove All Items, Remove Item, Select All Items, Set Item Selected, Toggle Item Select(ed).

List Wheel

Property Name	Type	Description	Related Event Actions
Alignment	-	Sets horizontal text alignment. Choices are: { Left | Center | Right }.	Set Item Alignment
Icon Position	-	Position of icons relative to text. Choices are: { LeftOf | RightOf }.	Set Icon Position
Icon Margin	Integer	Sets the space between icon and text. In pixels.	Set Icon Margin
Selected Index	Integer	Selects the default list item.	Set Selected Index
List Configuration Dialog	-	Defines the image/text for each entry in the list.	Set Item Icon, Set Item Icon (actually sets item text) Additional Related Event Actions: Append Item, Insert Item, Remove All Items, Remove Item, Select Next Item, Select Previous Item.

Panel – No additional properties.

Pie Chart Widget

Property Name	Type	Description	Related Event Actions
Start Angle	Integer	The starting angle of the pie chart in degrees.	Set Start Angle
Center Angle	Integer	The center angle of the pie chart in degrees. A positive value draws the chart counter-clockwise. Clockwise if negative.	Set Center Angle
Labels Visible	Boolean	Shows/Hides the labels for each data	Show/Hide Labels
Labels Offset	Integer	The position of the labels relative to the center of the pie chart, in pixels.	Set Label Offset
String Set	String Asset	The string asset containing the numeric characters for the tick labels. The asset must contain the characters for numbers 0 to 9.	Set Label String ID
Data Configuration Dialog	(See Description)	The Data Configuration Dialog lets users add data entries to the pie chart. The following properties can be set: • Value – Integer. The value of the entry • Radius – Integer. The radius, in pixels, of the pie for the entry • Offset – Integer. The offset, in pixels, of the pie from the center • Scheme – The color scheme for the ticks	None

Progress Bar

Property Name	Type	Description	Related Event Actions
Direction	-	Direction of progress bar. Choices are: { Right | Down | Left | Up }.	Set Direction
Value	-	Default value of the progress bar. The primitives `laProgressBarWidget_GetValue` and `laProgressBarWidget_GetValue` can be used to manipulate the widget's value during run time.	Set Value

Radial Menu Widget

Property Name	Type	Description	Related Event actions
Ellipse Visible	Boolean	Show the elliptical track of the widget	Elliptical track gets draw in Harmony Composer simulation and at runtime.
Highlight Prominent	Boolean	Highlights the prominent item when the widget rotation has completed its reset to the static, selectable position by drawing a rectangle behind the prominent item.	-
Ellipse Type	Enum	Selects the type of elliptical track Default – an elliptical track that best fits the widget area based on the size of the tallest and widest images with the size scale settings factored-in. Orbital – a “flatter” elliptical track that is best used with the Theta setting for a tilted look Rolodex – a vertical track with Theta setting locked at 90 degrees	Locks Theta to 90 degrees when Rolodex is selected
Theta	Integer	The angle (in degrees) of tilt relative to the y-axis of the ellipse. The number range is 0 to 90 degrees.	This field is only valid for Default and Orbital Ellipse Type setting. It is locked at 90 when Rolodex is selected.
a	Integer	This is the half-length (in pixels) of the 0-180 axis of ellipse. It is auto-calculated based on the widget size, the tallest image’s height, the ellipse type and scale settings.	-
b	Integer	This is the half-length (in pixels) of the 90-270 axis of ellipse. It is auto-calculated based on the widget size, the widest image’s width, the ellipse type and scale settings.	-
Size Scale Configuration • Size Scale	Enum	Off – all images displays at its original size Gradual – images in the very back are scale to the Minimum Size Modifier setting, the scale is gradually increased, with the prominent front item scaled to the Maximum Size Modifier setting Prominent – the image that is at the front, prominent location is scaled based on the Maximum Size Modifier, all other images are scaled to the Minimum Size Modifier setting	-
* Minimum Size Modifier	Integer	The value (in percent) for the widget to resize the image to. When Size Scale is set to Gradual, this value represents the lowest scale for the item in the back. When Size Scale is set to Prominent, this value represents the scaling value for every image in the widget except for the prominent item. This value is equal to or less than the Maximum Size Modifier value	-
* Maximum Size Modifier	Integer	The value (in percent) for the widget to resize the image to. When Size Scale is set to Gradual, this value represents the largest scale for the item in the front (prominent position). When Size Scale is set to Prominent, this value represents the scaling value for the prominent item. This value is equal to or greater than the Minimum Size Modifier value	-

Item List Configuration • Total Number of Items Shown * Total Number of Widget Items * Widget Items Configuration Dialog	Integer Integer (See Description)	The number images visible on the radial menu. This number does not may be less than or equal to the total images in the widget. The total number of images the widget contains. The Widget Items Configuration Dialog lets users add images to the widget. The follow properties can be set: • Image – Image Asset. The image to show for the widget item	The widget automatically space-out the images along the elliptical track base on this value. If this number is greater than Total Number of Items Shown, some of the images will be hidden in a FIFO queue in the back -
Touch Area Configuration • Show Touch Area * Touch Area X Offset * Touch Area Y Offset * Touch Area Width Percent * Touch Area Height Percent	Boolean Integer Integer Integer Integer	Show visually in Harmony Graphics composer the rectangular area that permits touch interaction. The X-coordinate in local space of the touch-allowed area for the widget. This is auto-calculated based on the Touch Area Width Percent. The Y-coordinate in local space of the touch-allowed area for the widget. This is auto-calculated based on the Touch Area Height Percent. The percentage of the width of the touch-allowed area as compared to the entire widget area. The percentage of the height of the touch-allowed area as compared to the entire widget area. The default value is 50.	This setting is for preview in Harmony Graphics composer only. The touch area is not rendered at runtime. - - If this value is less than 100 percent, the area is horizontally centered. If this value is less than 100 percent, the area is defined starting from the bottom of the widget.

Radio Button

Property Name	Type	Description	Related Event Actions
Text String	-	Select widget's text string from the Select String Dialog.	Set Text
Alignment: • Horizontal • Vertical	-	Text string alignment within the widget. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment
Group	Integer	Radio Button Group Number. Default is -1, indicating no group. Only one radio button in a group can have a default selected value of On. All others in the group are Off	N/A
Selected	Boolean	If selected, the button has a default value of On. All other buttons in the group have a Selected value of Off.	Select
Selected Image	-	Select image used for selected state. Default: no image.	Set Selected Image
Unselected Image	-	Select image used for unselected state. Default: no image.	Set Unselected Image
Image Position	-	Position of image relative to widget text. Choices are: { LeftOf | Above | RightOf | Below | Behind }.	Set Image Position
Image Margin	-	Space between radio button image and text, in pixels.	Set Image Margin
Circle Button Size	-	The diameter of the default circle button, in pixels	Set Circle Button Size

Rectangle

Property Name	Type	Description	Related Event Actions
Thickness	Integer	Line thickness in pixels.	Set Thickness

Scroll Bar

Property Name	Type	Description	Related Event Actions
Orientation	-	Scroll bar orientation. Choices are: { Vertical | Horizontal }.	Set Orientation
Maximum	Integer	Maximum scroll value (minimum = 0.)	Set Maximum Value
Extent	Integer	Length of scroll bar slider, re scroll bar maximum value. Indicates the number of lines or size of window visible at each scroll setting.	Set Extent
Value	Integer	Initial scroll bar value.	Set Value, Set Value Percentage
Step Size	Integer	Step size value of scroll bar arrow buttons. (Min = 1, Max = 9999).	Set Step Size Additional Related Event Actions: Step Backward, Step Forward

Slider

Property Name	Type	Description	Related Event Actions
Orientation	-	Orientation of the slider. Choices are: { Vertical | Horizontal }.	Set Orientation
Minimum	-	Minimum slider value.	Set Minimum Value
Maximum	-	Maximum slider value.	Set Maximum Value
Value	-	Initial slider value.	Set Value, Set Value Percentage
Grip Size	-	Grip size of slider, from 10 to 9999, in pixels.	Set Grip Size Additional Related Event Actions: Step

Text Field

Property Name	Type	Description	Related Event Actions
Text String	-	Select widget's text string from the Select String Dialog.	Clear Text followed by Append Text
Alignment	-	Horizontal alignment. Choices are: { Left | Center | Right }.	Set Alignment
Cursor Enable	-	Boolean. Show blinking cursor while editing.	Set Cursor Enabled
Cursor Delay	-	Cursor delay in milliseconds. From 1 to 999,999.	Set Cursor Delay Additional Related Event Actions: Accept Text, Append Text, Backspace, Clear Text, Start Editing.

Touch Test – No dedicated properties.

Window

Property Name	Type	Description	Related Event Actions
Title String	-	Select widget's title string from the Select String Dialog.	Set Title
Icon Image	-	Select image used. Default: no image.	Set Icon
Image Margin	Integer	Space between icon and title, in pixels.	N/A

Layer Properties and Event Actions

The property list for a graphic layer is close in look and feel to that of a widget. Each Layer has three property sets: Editor (see above), Widget (see above), and Layer (see below).

Layer Properties

Property Name	Type	Description	Related Event Actions
Transparency Enabled	Boolean	Automatically mask out pixels of with a specified color. If enabled Specify:	N/A
Mask Color	Integer	Red/Green/Blue or Red/Green/Blue/Alpha color value	N/A
All Input Passthrough	Boolean	Allow input events to pass through this layer to layers behind it.	N/A
VSync Enabled	Boolean	Layers should swap only during vertical syncs.	N/A
Buffer Count	Integer	Integer number of frame buffers associated with this layer, either 1 or 2.	N/A
Buffer N	-	For each buffer (N= 1 or 2) you specify:	-
Allocation Method	-	Buffer allocation method. Choices are: { Auto | Address | Variable Name } • Auto – Automatically allocate frame buffer space • Address – Specify a memory address • Variable Name – Use variable name as buffer location	N/A
Memory Address	-	If Address is the allocation method, specify the raw (physical) memory address as a hexadecimal number.	N/A
Variable Name	String	If Variable name is the allocation method, specify the variable name as a string value.	N/A

Screen Properties and Events

The property list for a screen shares the Name and Size properties with Layers and Widgets but has these unique properties.

Screen Properties

Property Name	Type	Description	Related Event Actions
Orientation	-	Display orientation: 0, 90, 180, 270 Degrees. This can also be set using the Display Manager.	N/A
Mirrored	Boolean	Enables screen mirroring.	N/A
Layer Swap Sync	Boolean	Enables that all layer buffer swapping happen at the same time, delaying lower layers until higher layers are finished drawing as well. For example, assume you make changes to layer 0 and layer 1 and you want to see those changes show up on the screen at the same time. Without this option you'd see layer 0's changes as soon as it finishes when layer 1 has not yet started drawing. This option will hold layer 0's swap operation until layer 1 finishes as well. **Note:** Currently, this property is only supported by the CLCD Graphics Controller Driver and is ignored by all other drivers.	N/A
Persistent	Boolean	Indicates that the screen should not free its widgets and memory when it is hidden. This results in faster load times and persistent data, but at the cost of higher memory consumption.	N/A
Export	Boolean	Includes this screen the application build. This can also be set using the Screens panel.	N/A
Primary	Boolean	Sets this screen as the primary screen. The primary screen is the first screen displayed when the application starts. This can also be done using the Screens Panel Generate check box.	N/A

Graphics Composer Asset Management

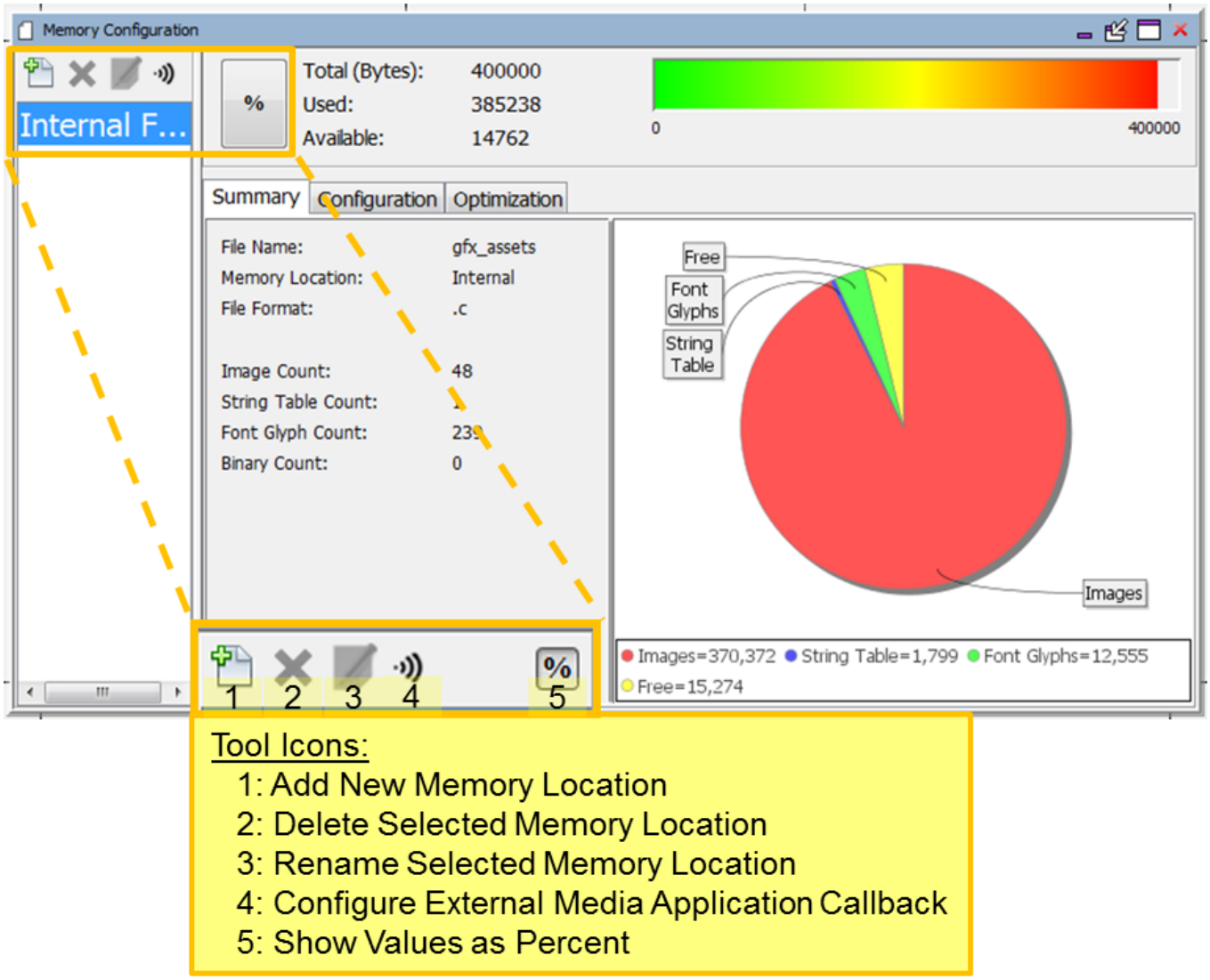
The Asset menu supports managing all graphical assets (memory, images, languages, fonts, strings, and binary data).

Memory Configuration

Provides information on configuring memory locations.

Description

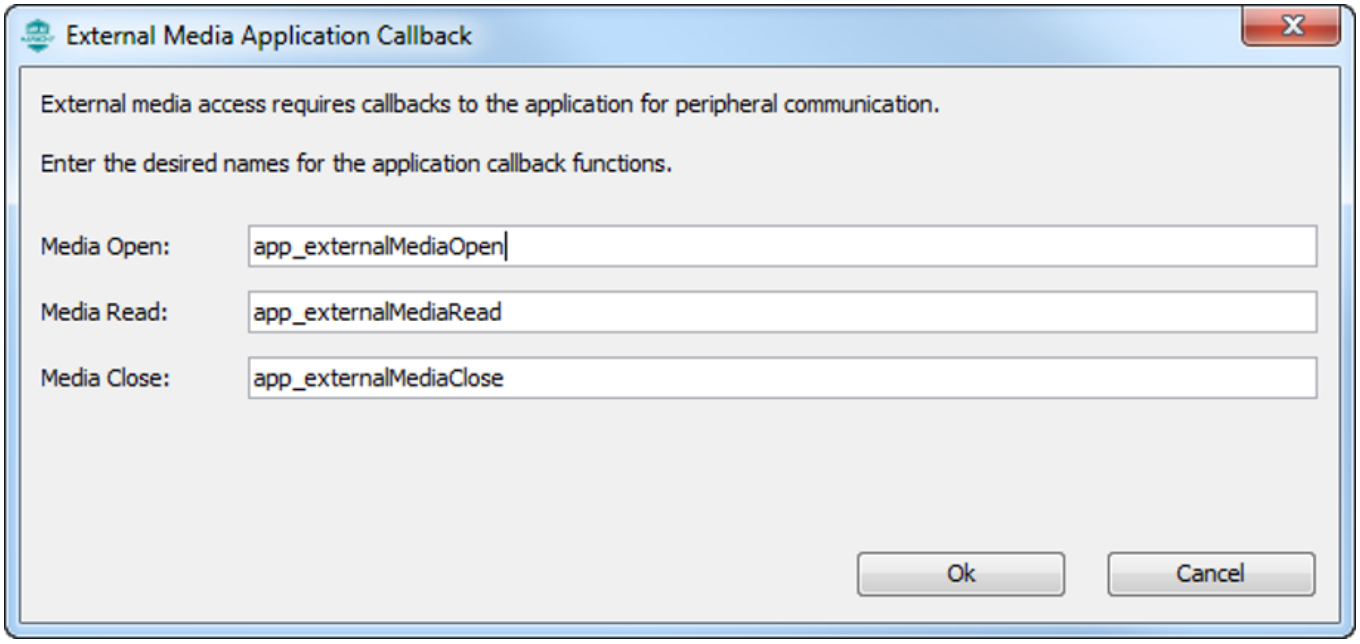
The Memory Locations window is launched from the Graphics Composer's Asset menu. Selecting Memory Locations this brings up a window with three sub-tabs (in this example, the Aria Showcase demonstration is referenced):



Window Toolbar

The window's tools icons support:

1. **Add New Memory Location** – This supports multiple external memory resources.
2. **Delete Selected Memory Location** – Removes a previously defined memory location.
3. **Rename Selected Memory Location** – Renames a previously defined memory location.
4. **Configure External Media Application Callback** – This allow definition of media callbacks, which must be provided in the project.



5. **Show Values as Percent** – Memory utilization on the bar graph can be in bytes or as a percent of the total internal flash memory assigned to support asset storage. (That memory allocation is set using the Configuration sub-tab.)

The APIs for the external media callback functions are as follows:

```
GFX_Result app_externalMediaOpen(GFXU_AssetHeader* asset);
GFX_Result app_externalMediaRead(GFXU_ExternalAssetReader* reader,
                                 GFXU_AssetHeader* asset,
                                 void* address,
                                 uint32_t readSize,
                                 uint8_t* destBuffer,
                                 GFXU_MediaReadRequestCallback_FnPtr cb);
void app_externalMediaClose(GFXU_AssetHeader* asset);
```

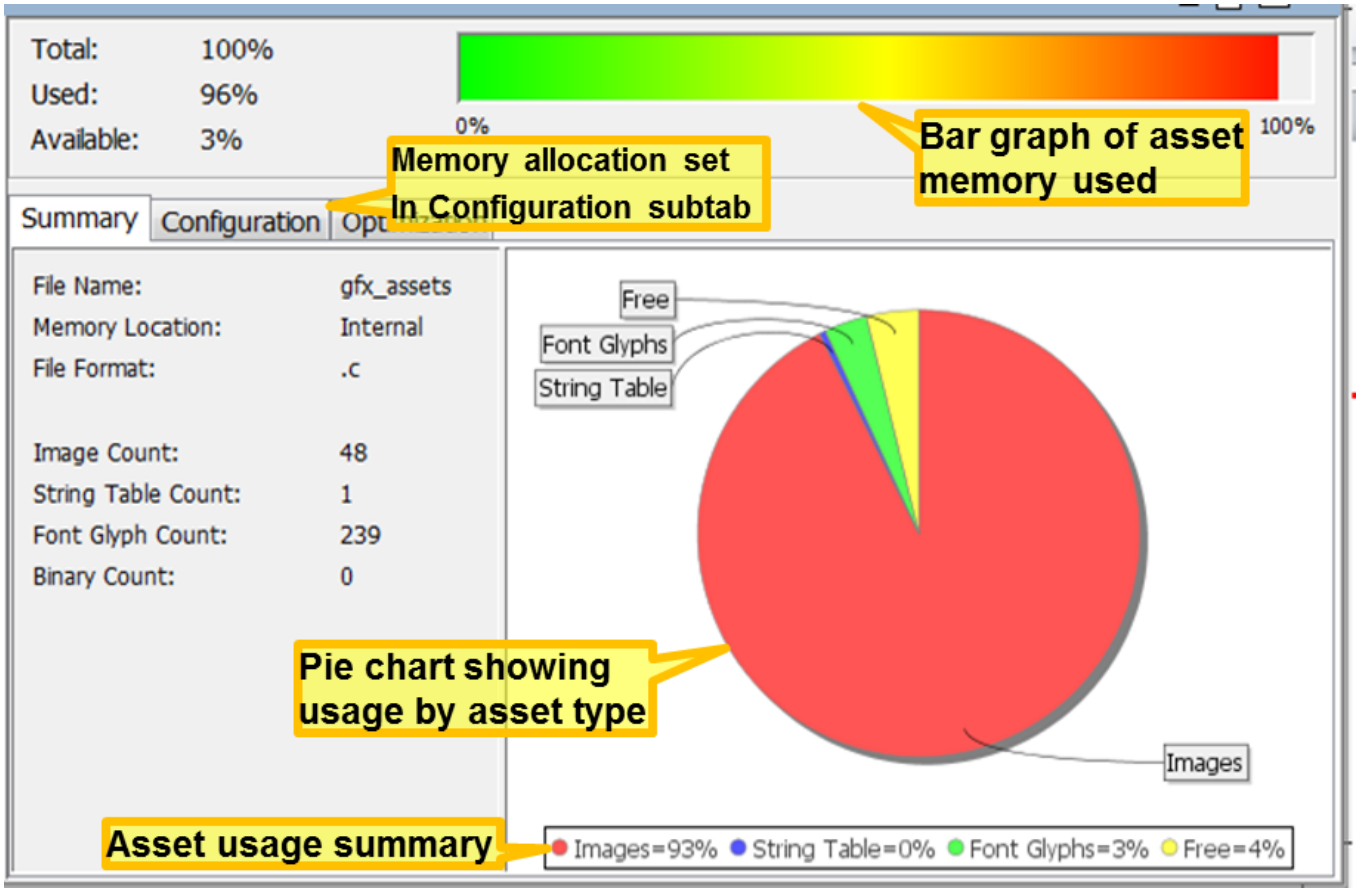
The graphics demonstration project, aria_external_resources, provides an example of how to write these callbacks. This demonstration supports three types of external memory: SQI External Memory, USB Binary, and USB with File System. Examples of these callbacks are found in the project's `app.c` file. The Aria demonstration projects Aria External Resources and Aria Flash provide more details on how to use external memory to store graphics assets.

Sub-tabs

There are three sub-tabs to this window.

Summary Sub-tab

This sub-tab summarizes program flash allocations for images, strings, and fonts.

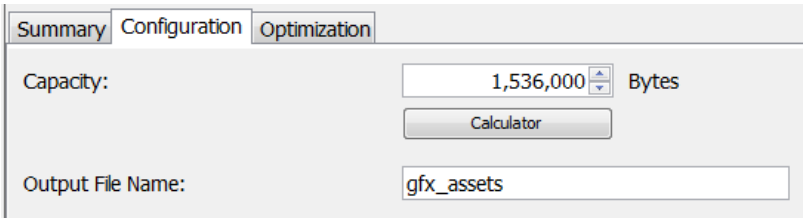


The memory allocation shown for “Font Glyphs” measure the space that holds all the font glyphs used by the application, either by static strings or by glyph ranges defined in support of dynamic strings. Strings are defined by arrays of pointers to glyphs, so string memory usage measures the size of these arrays, not the actual font glyphs used. (“Glyph” is defined here.)

Note: The word “glyph” comes from the Greek for “carving”, as seen in the word hieroglyph – Greek for “sacred writing”. In modern usage, a glyph is an elemental or atomic symbol representing a readable character for purposes of communicating through writing.

Configuration Sub-tab

This sub-tab specifies the intended allocation of internal (program) flash memory to graphics assets (Total Size). (The default value is 1024 bytes.) It also names the graphics assets file name (here it will be `gfx_assets.c`). The allocation of flash is only used to scale the Total/Used/Available bar graph at the top of the display. Under sizing or oversizing this amount does not affect how the application is built.

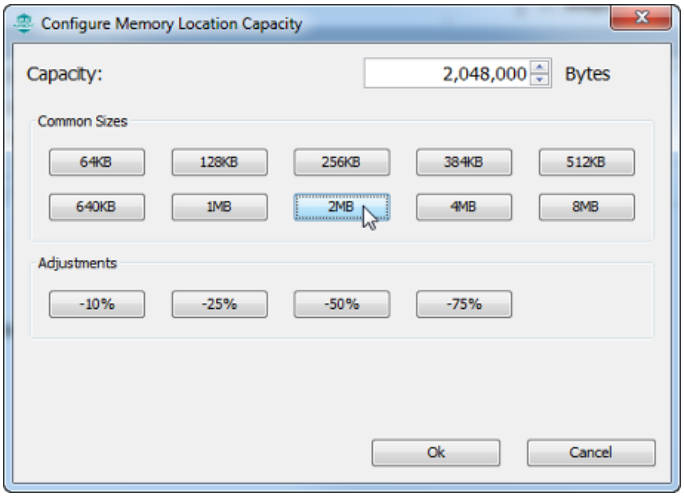


If your device has 1024 Kbytes (1048576 bytes) of flash, you can assign 40% to asset storage and 60% to code. In that case the “Total Size” in the above sub-tab would be set to 419430 (= 40% of 1048576).

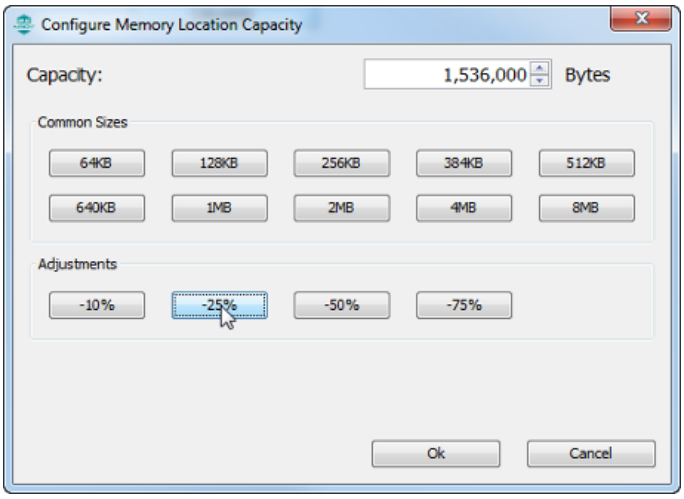
The Calculator button can assist you in allocating internal flash. Click on it and then set the device flash capacity. Then you can apply an adjustment to that value to assign that memory to asset storage.

Example:

If the device has 2 Mbytes of internal Flash, click **2MB**.



Then, to assign 75% of the 2 Mbytes to asset storage, click **-25%** to reduce the 2 MB by 25%, leaving 75%, and then click **OK** to finish. This will then assign 1,536,000 bytes to asset storage.

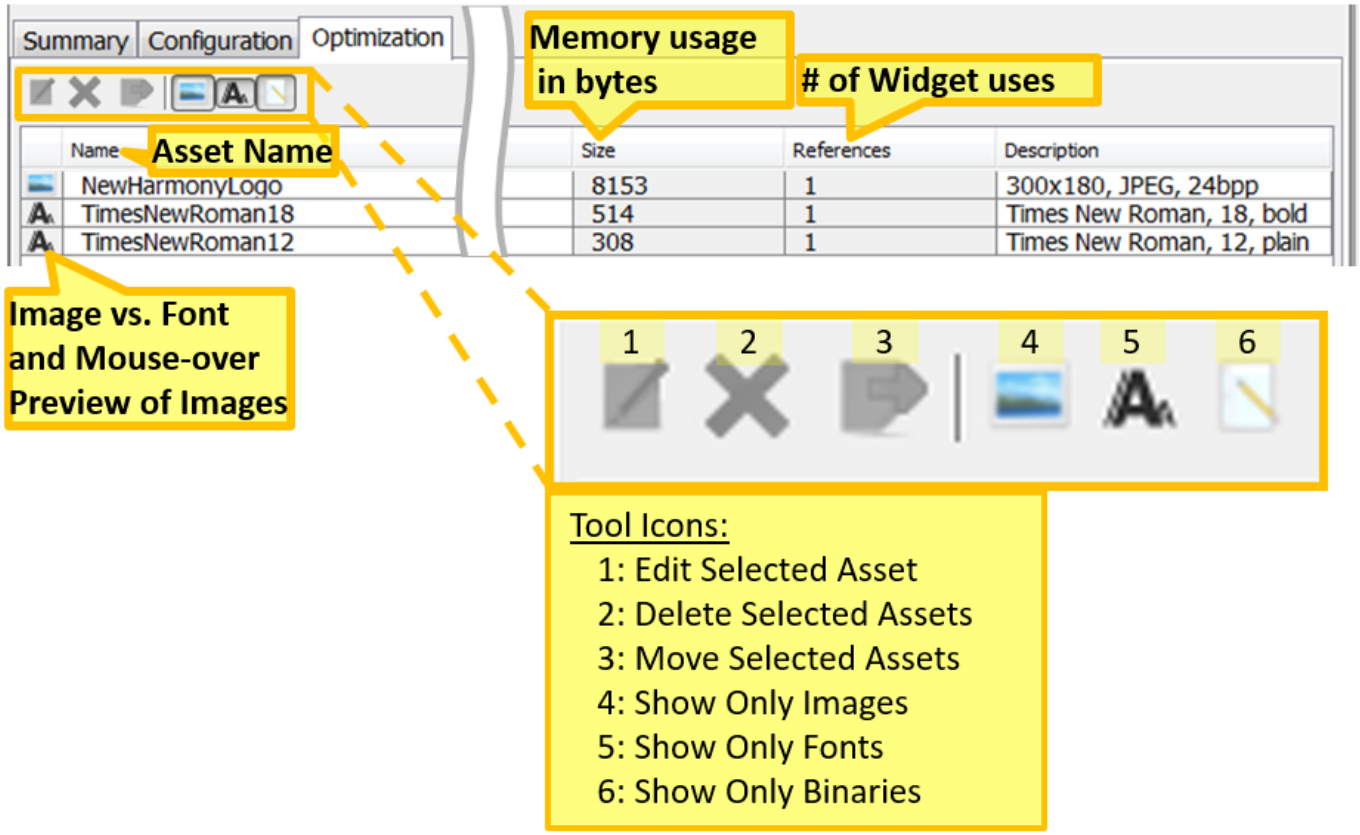


Internal (program) Flash is shared between the application's code and asset storage. If the application code and graphics assets (fonts, strings, images) won't fit into the available flash memory then the linker will be unable to build the application and an error will be generated in MPLAB X IDE.

The **Output File Name** must be compatible with the operating system hosting MPLAB X IDE. In most cases the default name (`gfx_asset.c`) will suffice, but this is provided for additional flexibility in building the application.

Optimization Sub-tab

The Optimization sub-tab for the Aria Quickstart demonstration is shown in the following figure.



The **Size** column shows the bytes allocated for storage in internal flash for the images, fonts, and binaries of the application.

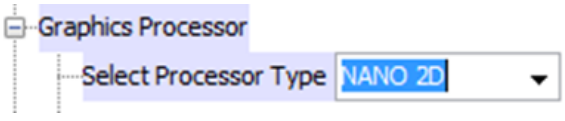
The **References** column shows the number of known references for these assets by the application's widgets. A references count of zero suggests that the asset is not used by the application, but it could also mean that the asset is only used in real-time when it is dynamically assigned to a widget by the application. Clicking the title of a column (Name, Size, or References) sorts the lists of graphics assets by that column. Clicking the same column again reverses the sort order.

The window's tools icons support:

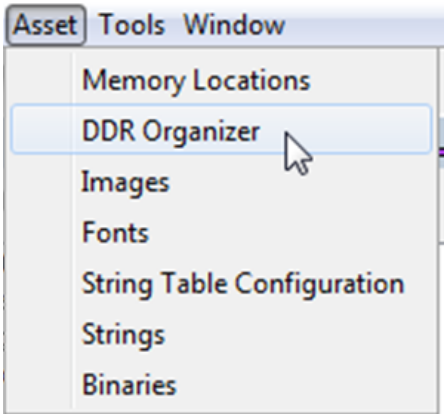
1. **Edit Selected Asset** – This brings up the edit dialog for the image, font, or binary chosen
2. **Delete Selected Assets** – Removes the selected assets
3. **Move Selected Assets** – Move assets from one location to another. This is useful for moving assets to/from internal memory from/to external memory.
4. **Show Only Images** – Show image assets toggle on/off
5. **Show Only Fonts** – Show font assets toggle on/off
6. **Show Only Binaries** – Show binary assets toggle on/off

DDR Organizer

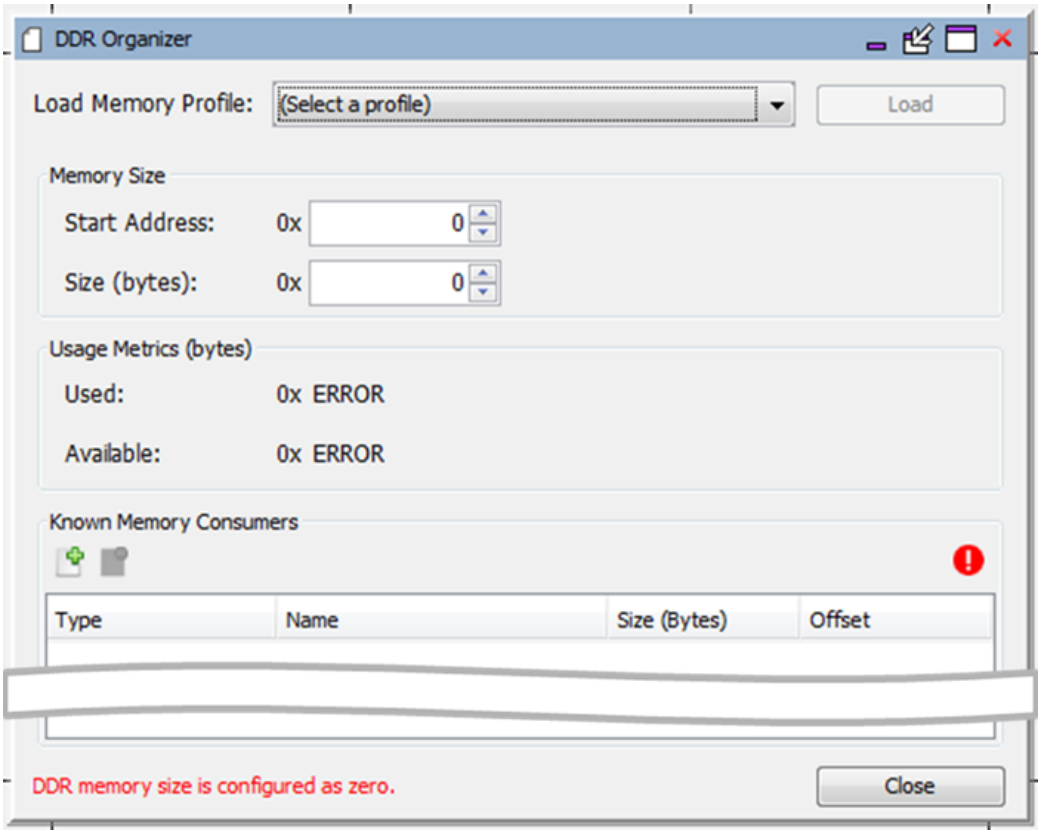
The DDR Organizer tool supports managing buffers, raw images, and other memory resources in the DDR memory of DA devices and only DDR-enabled DA devices. This tool also requires that the DA's built-in 2D graphics processor be enabled. Under *Harmony Framework Configuration* > *Graphics Stack* > *Graphics Processor*, select the **NANO 2D processor**:



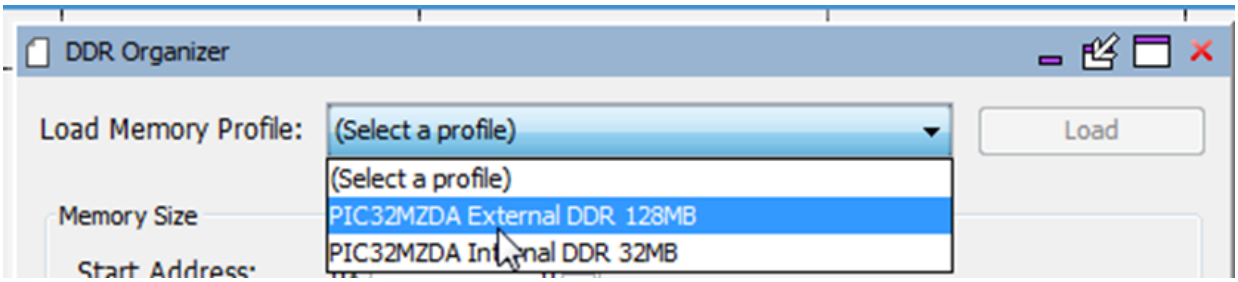
The DDR Organizer tool is launched from the Assets Management pull-down menu:



The following window will appear if the tool has not been used before for the active project target configuration:

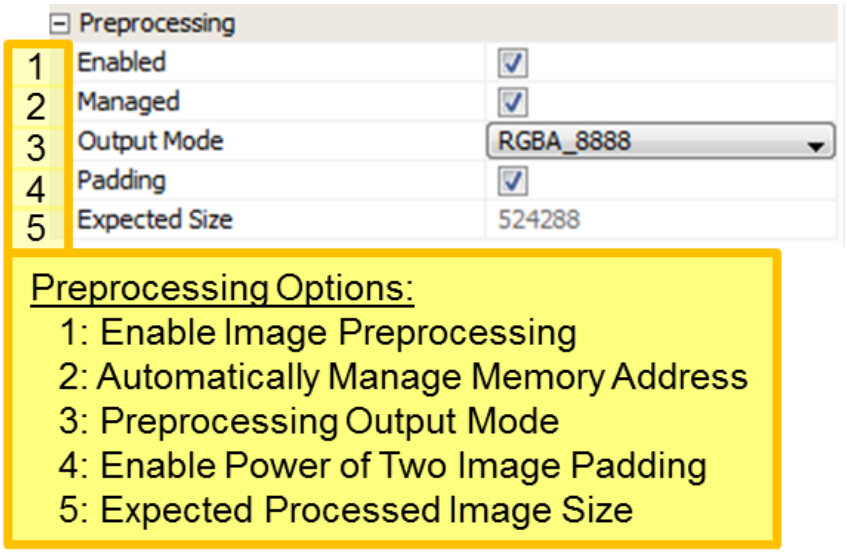


Select the memory profile that corresponds to the target DDR-enabled DA device:

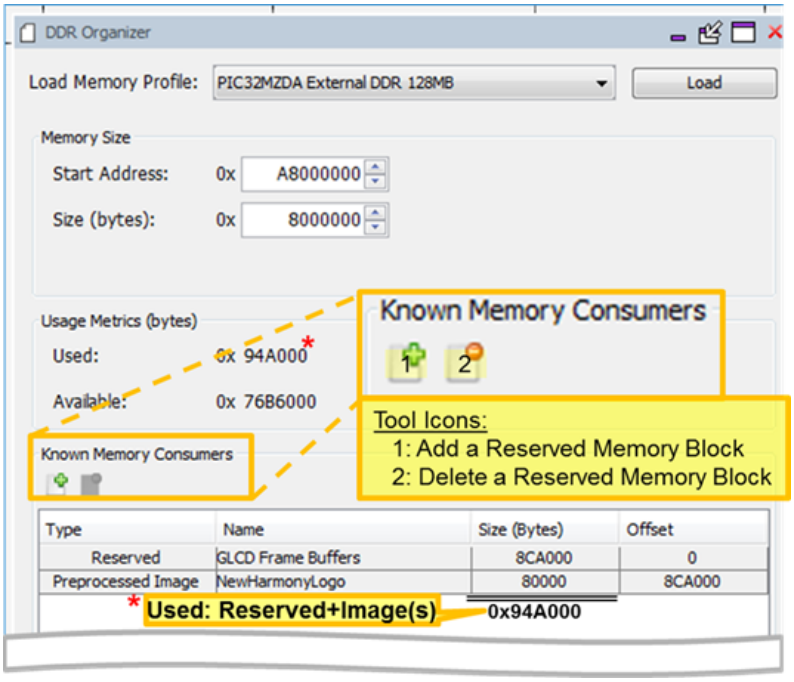


Then select the **Load** Button to load that memory configuration into the tool.

When Preprocessing is enabled for an image under the Image Assets tool:



An entry for the image appears in the DDR Organizer window:



When the memory profile is loaded, the tool automatically reserves DDR memory for the GLCD Frame Buffers sufficient for three double-buffered layers, allocating 32 bits (4 bytes) for RGBA_8888 format for each pixel. This provides 384,000 pixels (800x480) per frame buffer.

The tool icons support adding non-image memory allocations to the DDR memory map. To add or remove the memory allocation belonging to an image, the Preprocessing enabled property for that image is enabled/disabled using the Image Asset tool.

Image Assets

Provides information on the Image Assets features.

Description

The Image Assets window is launched from the Graphics Composer's Asset menu.

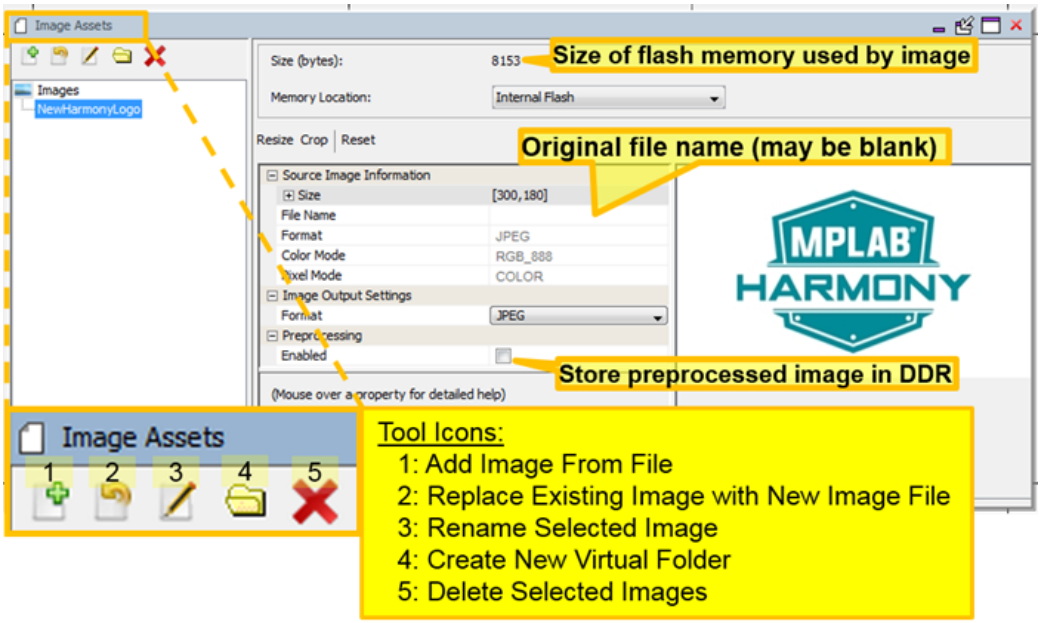
The Image Assets window lets you import images, select different image formats/color modes for each image, select compression methods (for example, RLE) for each image, and displays the memory footprint of each. Images can be imported as a BMP, GIF, JPEG, and PNG (but not TIFF). Images can be stored as Raw (BMP, GIF), JPEG, and PNG.

Note: MHGC does not support image motion that can be found in GIF (`.gif`) files. GIF images are stored in the raw image format, meaning that there is no image header information stored with the image.

When an image is imported into MGHC, the Graphics Asset Converter (GAC) stores the input format and color mode along with any relevant header data. The image's pixel data is then promoted from its native format into a Java Image using 32 bits/pixel (8 bits for each color, RGB, and 8 bits for Alpha Blending). If the image contains Alpha Blending then this information is stored in the "A" of RGBA, otherwise the "A" is set to maximum opacity. When the application is built each image is stored in the image format and color mode selected. Images displayed in the Screen Designer are converted from Java Image format into the format/color mode selected so that the Screen Designer accurately represents what the application will show when running.

The images are decoded on the fly by the graphics library and rendered on the screen. This provides the designer with considerable flexibility to import using one format and store resources using another format, thus exploring and maximizing the best memory utilization for their application and hardware. This supports trading a smaller memory footprint at the cost of additional processing (for static or drawn-once) or reducing processing at the cost of a larger memory footprint (dynamic or drawn many times).

The following figure shows the Image Assets window for the Aria Quickstart demonstration.

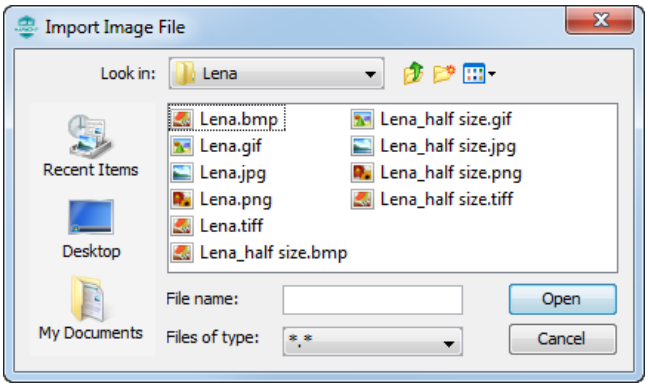


Window Toolbar

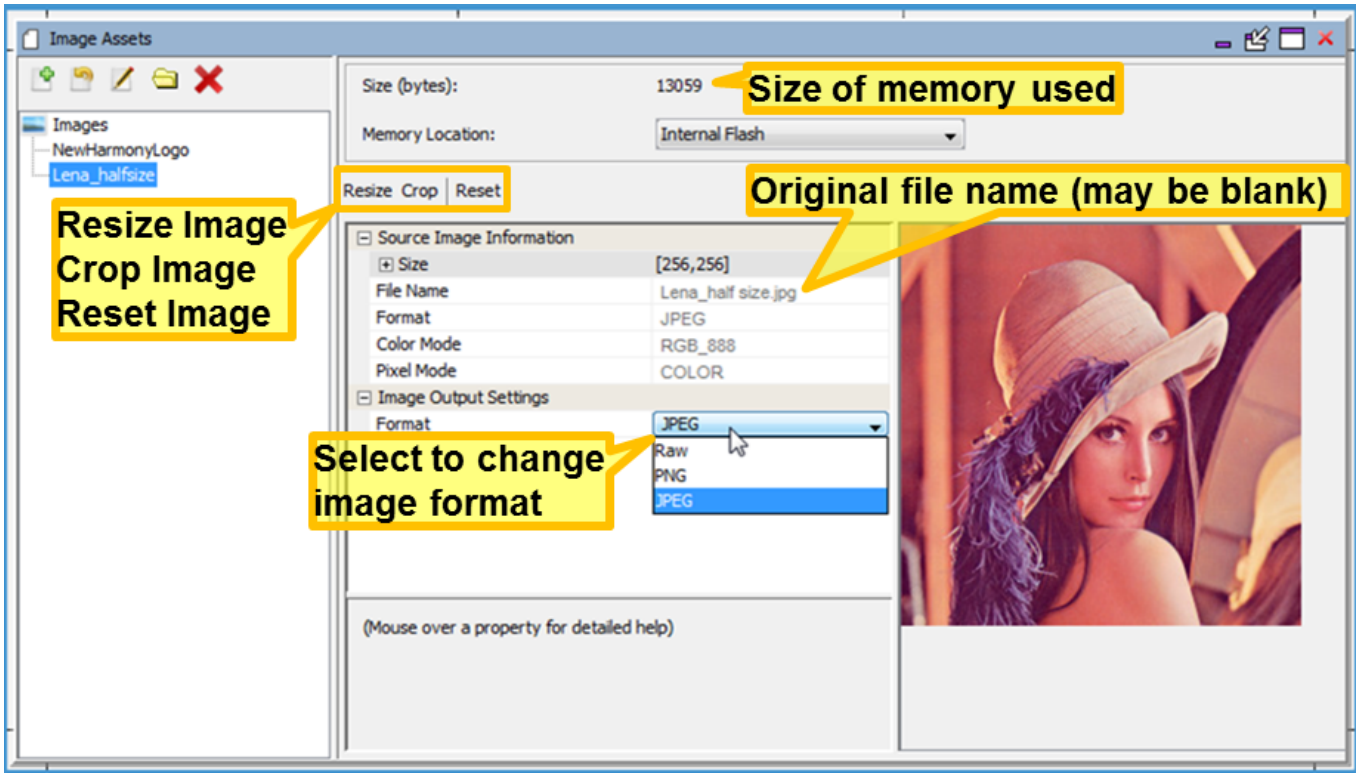
There are five icons on the toolbar below the Images tab:

1. **Add Image Asset** – Brings up "Import Image File" dialog window to select image file to add to the graphics application.
2. **Replace Existing Image with New Image File** – Brings up the same "Import Image File" dialog but instead of creating a new image, the file's content replaces the currently selected image.
3. **Rename Selected Image** – Renames the selected image.
4. **Create New Virtual Folder** – Creates a new virtual folder, allowing you to organize images in a hierarchy.
5. **Delete Selected Images** – removes the selected images from the application.

Selecting the Add Image Asset or Replace Existing Image icon opens the Import Image File dialog that can be used to select and import an image.



After selecting the file and clicking **Open**, the Image Assets window opens.



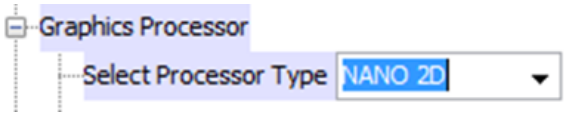
The size of the memory used for this image based on its color mode, format, compression, and global palette usage is shown by Size (bytes). See Image Format Options below for more details.

The File Name of the original source file is also shown, but may be blank if the image was imported under MPLAB Harmony v2.03b or earlier. The format and color mode of the stored image can be changed to reduce the image's memory footprint. (If using an LCC controller, you can also turn on the Global Palette, replacing each pixel in the image with just an 8 bit LUT index.)

The three internal image formats are:

- Raw – binary bit map with no associated header information. GIF and BMP images are imported into this format.
- PNG – lossless image format with compression, 24 bits/pixel (RBG_888) or 32bits/pixel (RGBA_8888). A good choice for line drawings, text, and icons.
- JPEG (JPG) – loss compressed format, uses much less storage than the equivalent bit map (raw). Good for photos and realistic images.

New to Harmony 2.06 is the option to preprocess an image into raw pixels at boot-up, which will greatly improve image draw/redraw times though the use of the high performance 2-D graphics processing unit (GPU) that is available on DDR-enabled DA devices. Be sure that this feature is enabled in MPLAB Harmony Configurator. Under Harmony Framework Configuration > Graphics Stack > Graphics Processor, select the NANO 2D processor:

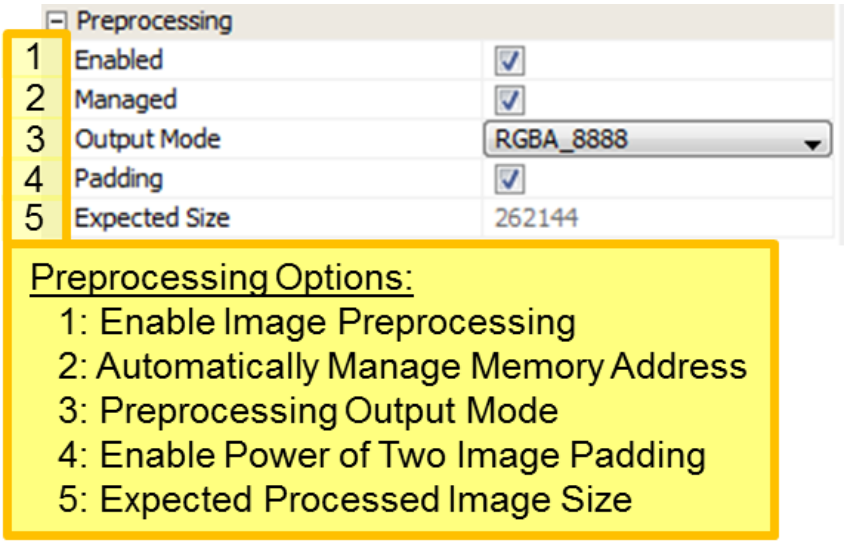


Note: Do not enable image preprocessing except on DDR-enabled DA devices with the NANO 2D graphics processor enabled. To do so will produce an application that builds but does not run.

With Preprocessing of the image enabled, additional options become available:

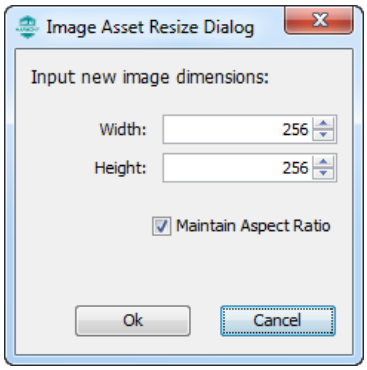
- DDR Memory allocation for the image is automatically handled when the **Managed** option is selected
- The **Output Mode** should be selected to match the GLCD's color mode, typically RGBA_8888
- The **Padding** option expands the image size to the nearest power of two. For example, a 480x212 image would be increased to 512x256 pixels.
- The expected size of the preprocessed image in DDR memory is shown in the **Expected Size** entry

For more information on how images are stored within DDR memory, see the section on the Asset Management DDR Memory tool above.

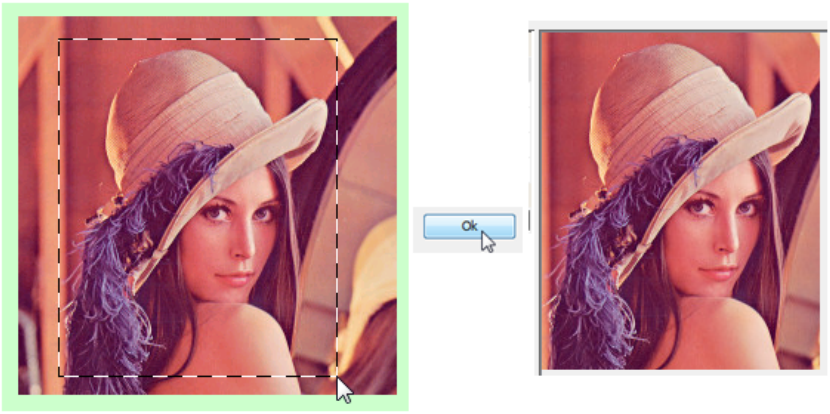


The Image Assets window supports resizing, cropping, or resetting an image:

- **Resize** – Brings up a dialog window to change the pixel dimensions of the image. The image is interpolated from the original pixel array into the new pixel array.



- **Crop** – Places a cropping rectangle on the image. Click and drag a rectangle across the image to select the new image. Then click **Ok** to crop the image.



- **Reset** – Allows undoing of a resize or crop. The original image is always stored in the project, so a Reset is always available to return the image to its original state.

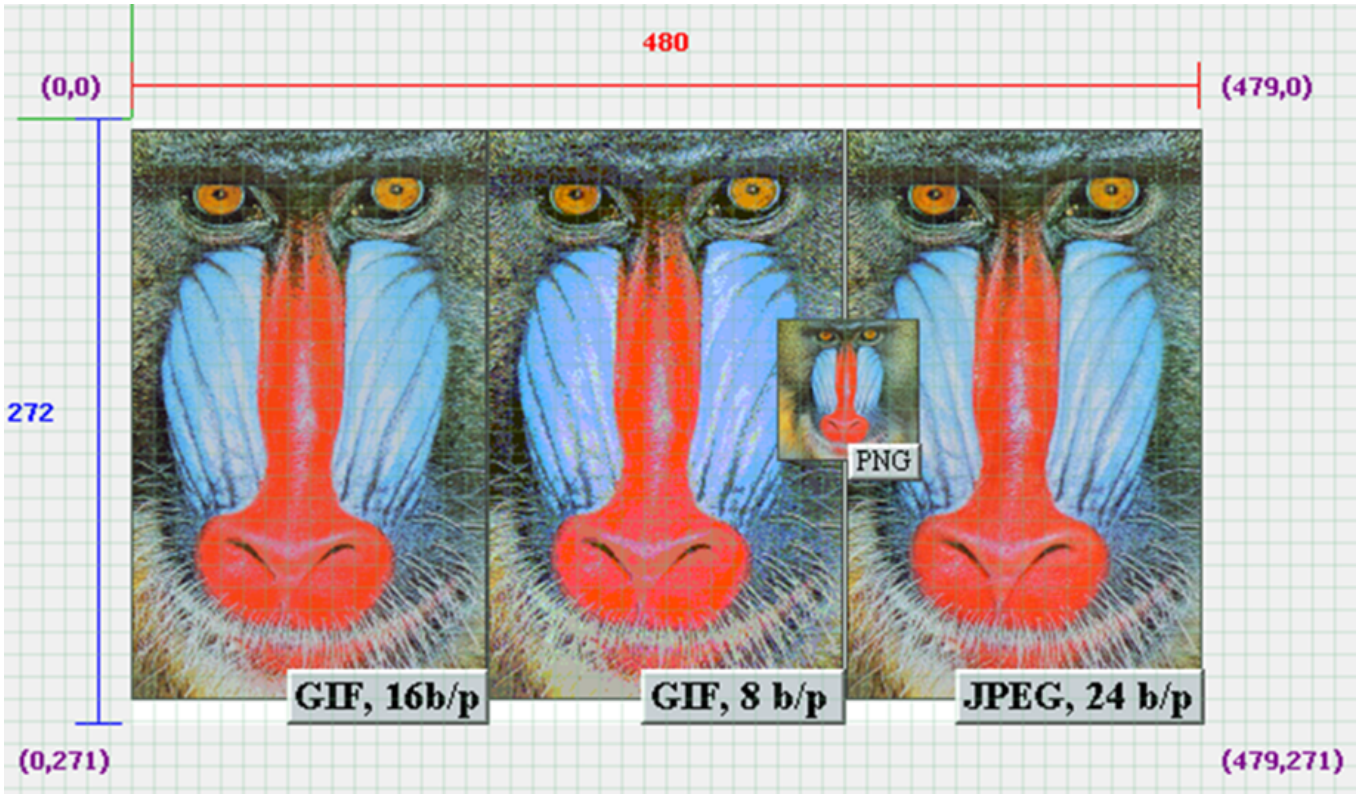
Original images are retained by MHGC by the superset Java Image format. So an image crop will change how the image is stored in the application but not how it is stored in MHGC. Reset will always restore the image back to the original pixels. (Reset is not an “undo”.)

Example Images

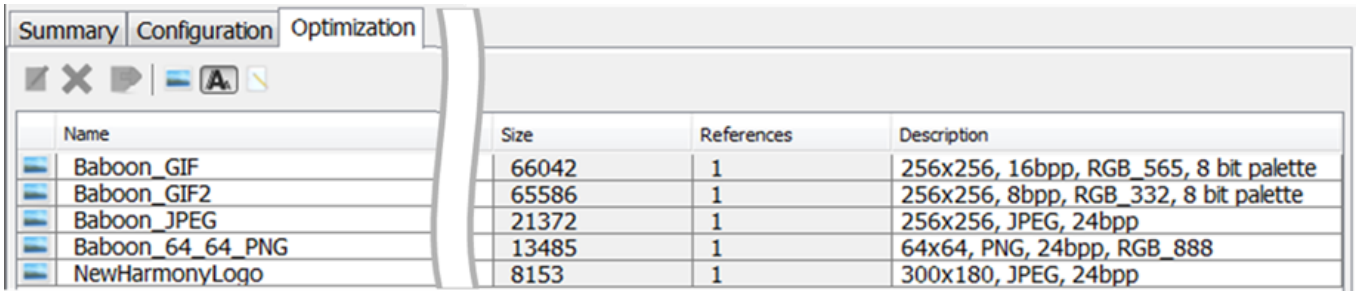
Example images are available from many sites on the internet. One of the best sites is found at the USC-SIPI Image Database (http://sipi.usc.edu/database/). There are many canonical test images, such as Lena, The Mandrill (Baboon), and other favorites, all in the TIFF

format. The TIFF format is not supported by the Graphics Composer, but you can easily convert from TIFF to BMP, GIF, JPEG, or PNG using the export feature found in the GNU Image Manipulation Program (GIMP), which is available for free download at: https://www.gimp.org. GIMP also allows you to change the pixel size of these images, usually 512x512, to something that will fit on the MEB II display (either 256x256 or smaller).

The following figure shows the Graphics Composer Screen Designer for the pic32mz_da_sk_meb2 configuration of the Aria Quick Start project after adding three images.

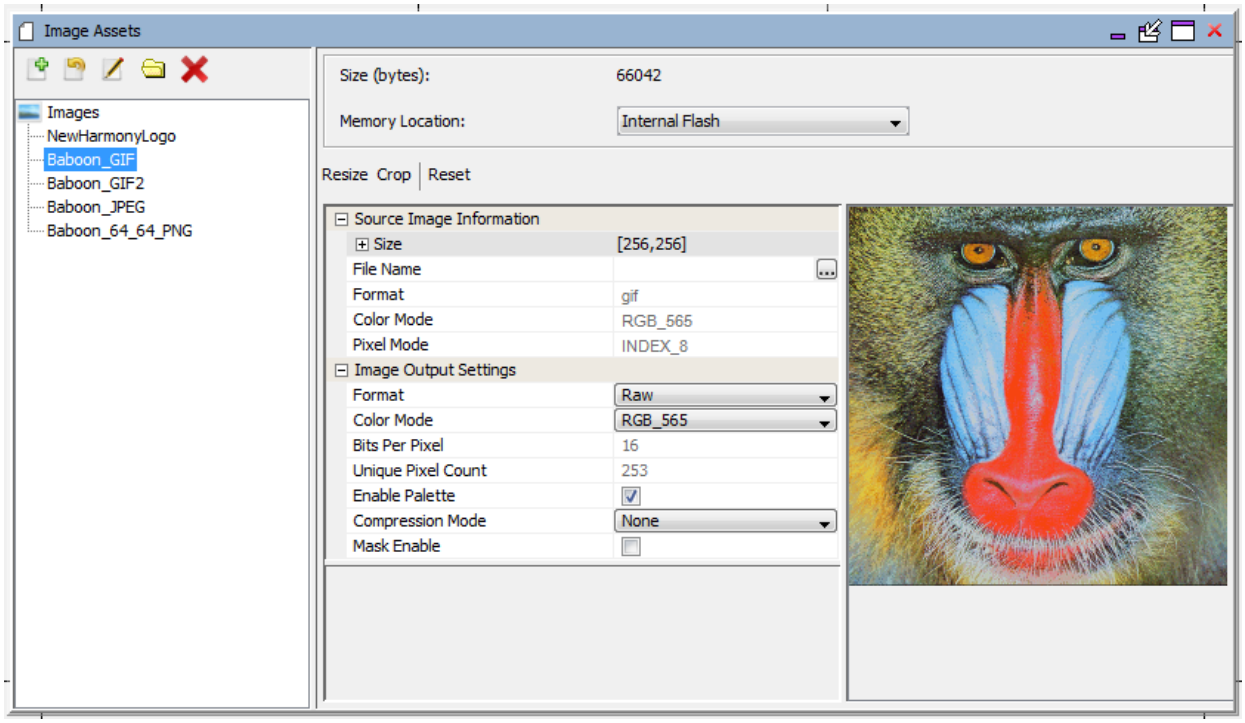


The following figure shows the Optimization Tab after adding these images.



Name	Size	References	Description
Baboon_GIF	66042	1	256x256, 16bpp, RGB_565, 8 bit palette
Baboon_GIF2	65586	1	256x256, 8bpp, RGB_332, 8 bit palette
Baboon_JPEG	21372	1	256x256, JPEG, 24bpp
Baboon_64_64_PNG	13485	1	64x64, PNG, 24bpp, RGB_888
NewHarmonyLogo	8153	1	300x180, JPEG, 24bpp

Selecting the Baboon_GIF image and the Edit Selected Asset icon ( ) opens an Image Assets window, as shown in the following figure.

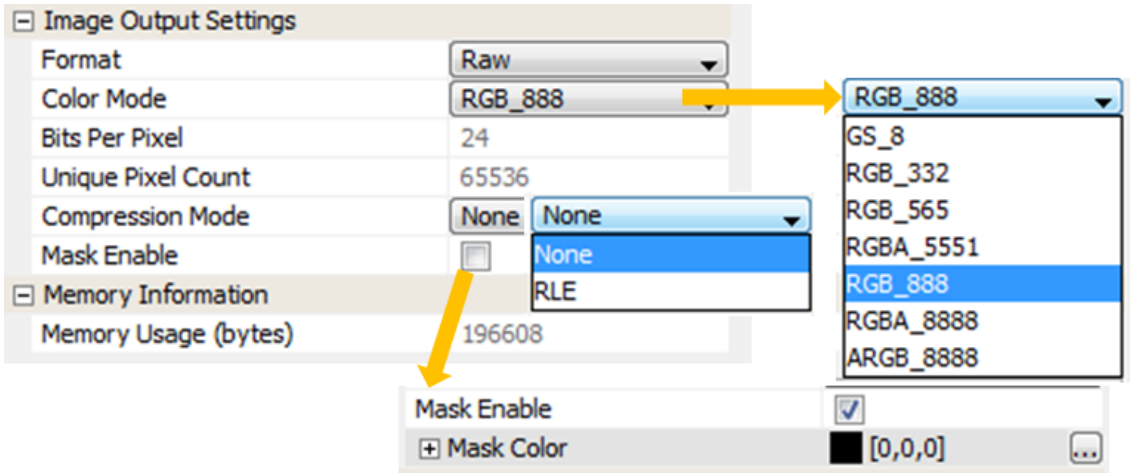


Because this image had only 253 unique pixel colors (Unique Pixel Count = 253) the Enable Palette option was automatically enabled. This feature, which works on an image by image basis, is separate from enabling a Global Palette. The image is stored using 8 bits of indexing into an image-specific lookup table (LUT). If the image has more than 256 unique colors then the Enable Palette option is not available and is not shown.

Image Format Options

Raw Format Images

Raw format images have the following options:



Regardless of the Color Mode of the imported the image, the stored image can be stored in a different color mode. For example, a JPEG image could be in 24 bits/pixel RGB format but stored in the application using RGB_565 or even RBG_332 to save space. The Project Color Mode (set through the *File* > *Settings* menu) is different from the Color Mode of images. This is determined by the capabilities of the projects graphics controller. The graphics library converts images from the stored color mode to the project's color mode before output.

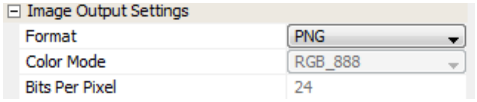
If the image has 256 or less unique pixel colors an option to Enable Palette is set by default. If the image has more than 256 unique colors this option is not displayed. This replaces the palette pixels with 8-bit indices into the image's palette look up table (LUT). NOTE: Enabling the Global Palette disables this for all images and all image pixels are replaced by 8-bit indices into the global palette LUT.

The Compression Mode for a raw format image is either None (no compression) or RLE for run-length encoding.

Image masking is a form of cheap blending. For example, given the following image, you may want to show the image without having to match the lime green background. With image masking you can specify that the lime green color as the "mask color", causing it to be ignored when drawing this image. The rasterizer will simply match a pixel to be drawn with the mask. If they match, the pixel is not rendered.

PNG Format Images

For PNG format images you can change the image format and the image color mode:



JPEG Format Images

For JPEG format images you can change from JPEG format to Raw or PNG:

Once changed from JPEG into another format, the new format will have other options.

Managing Complex Designs

The Image assets tool lists the images in the order of their creation. In a future version of MPLAB Harmony this will be sortable by image name. For now, it is recommended that you use the Memory Locations asset tool, and use the Optimization sub-tab instead to manage a complex set of images. The Optimization sub-tab allows you to sort graphics assets (fonts, images, binaries) by Name, Size, and number of widget References. This makes it much easier to find and edit an image by its name rather than order of creation.

Font Assets

Provides information on the Font Assets features.

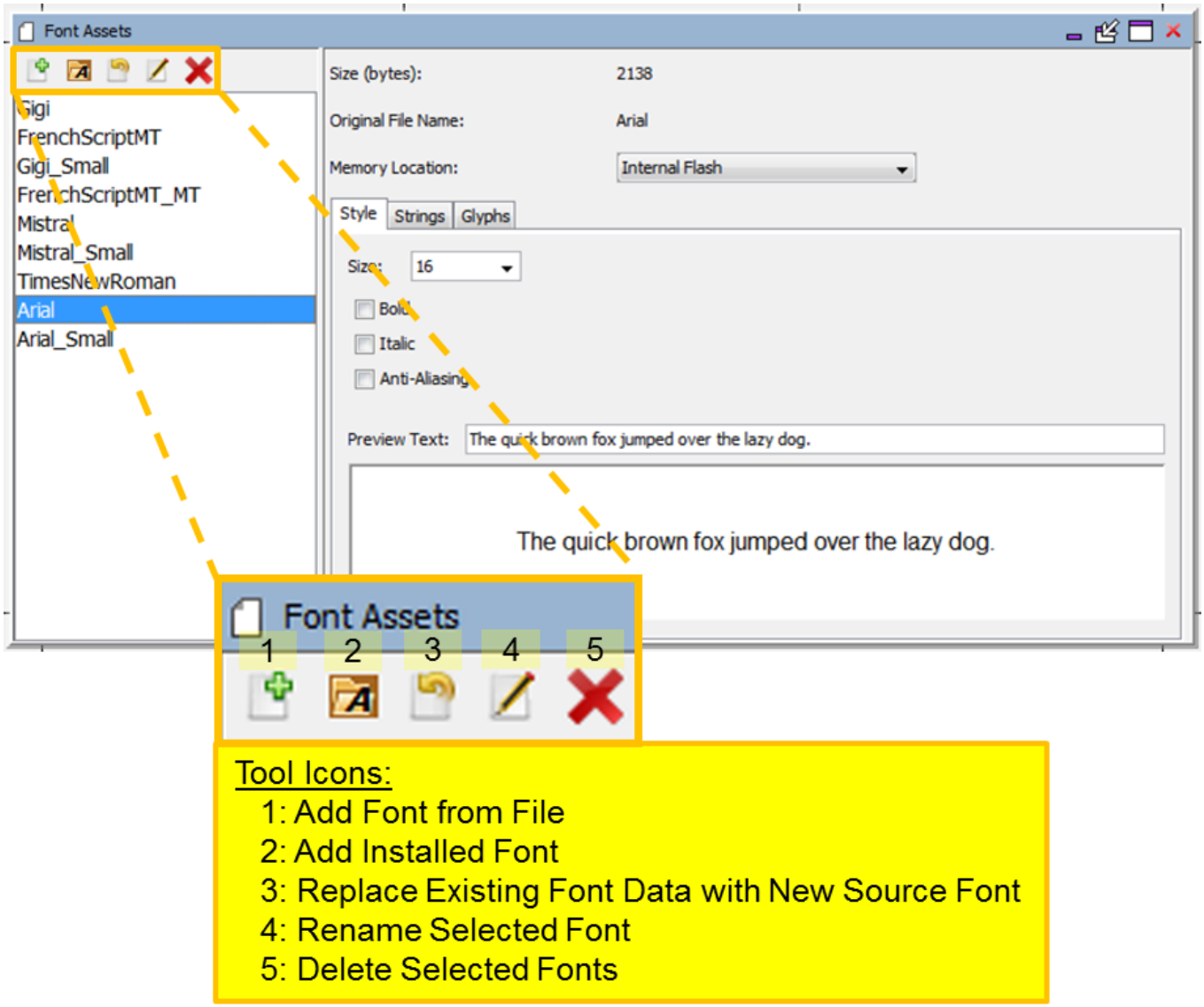
Description

The Font Assets window is launched from the Graphics Composer's Asset menu.

Note: There are three dimensions to text support: Languages, Fonts, and Strings. Language "ID" strings are identified when an application supports more than one language. (In the case of single language support, the language default is provided.) Fonts are imported and organized using the Font Assets window. Strings are defined by a string name, and this name is used by widgets to reference the string. For each string and each language supported the glyphs are defined to spell out the string's text and the font is chosen for that text.

- Languages are managed within the String Table Configuration window
- Fonts are managed within the Font Assets window (this topic)
- Strings are managed within the String Assets window

The following figure shows the Font Assets window from the Aria Coffee Maker demonstration.

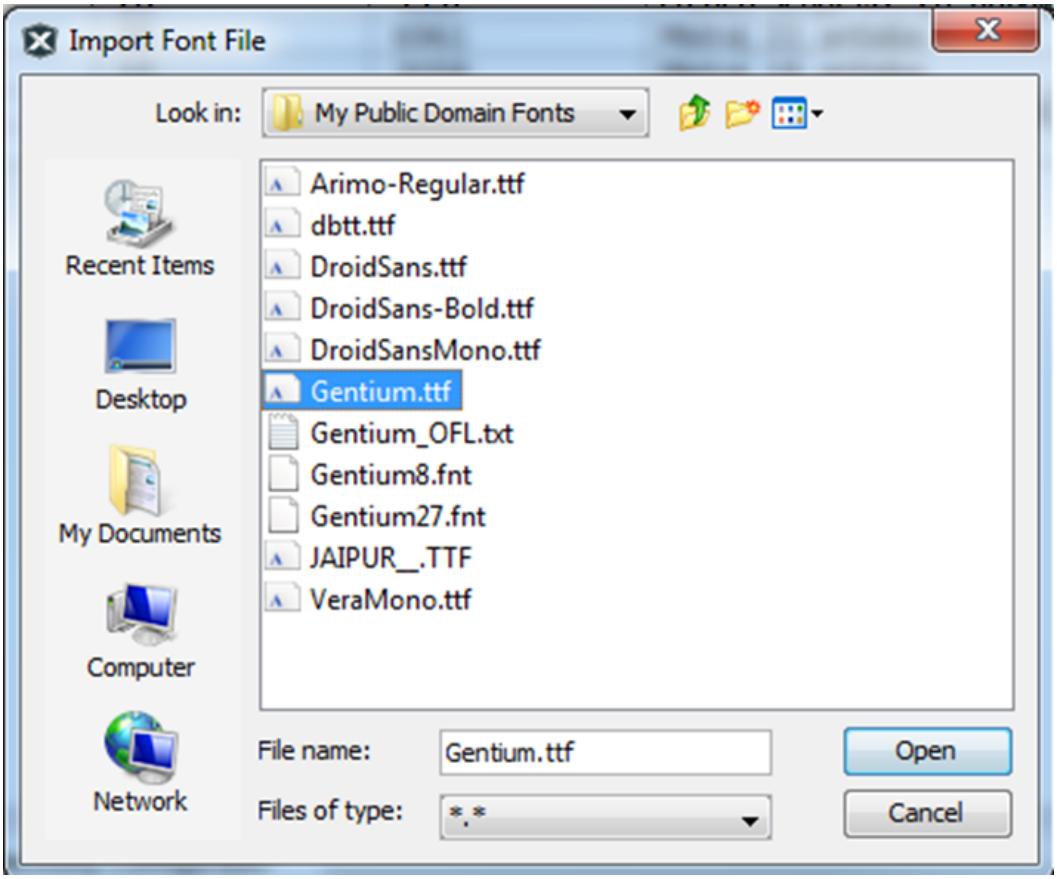


The Size (bytes): for a Font asset shows how much memory is needed to store all the glyphs used by the application from this font. For static strings MHGC determines which glyphs are used by the application's pre-defined strings and builds these glyphs into the application. For dynamic strings (i.e. strings created during run time) ranges of glyphs are selected by the designer and these ranges are also included in the application by MHGC. The memory needed to store all these glyphs is shown by Size (bytes): .

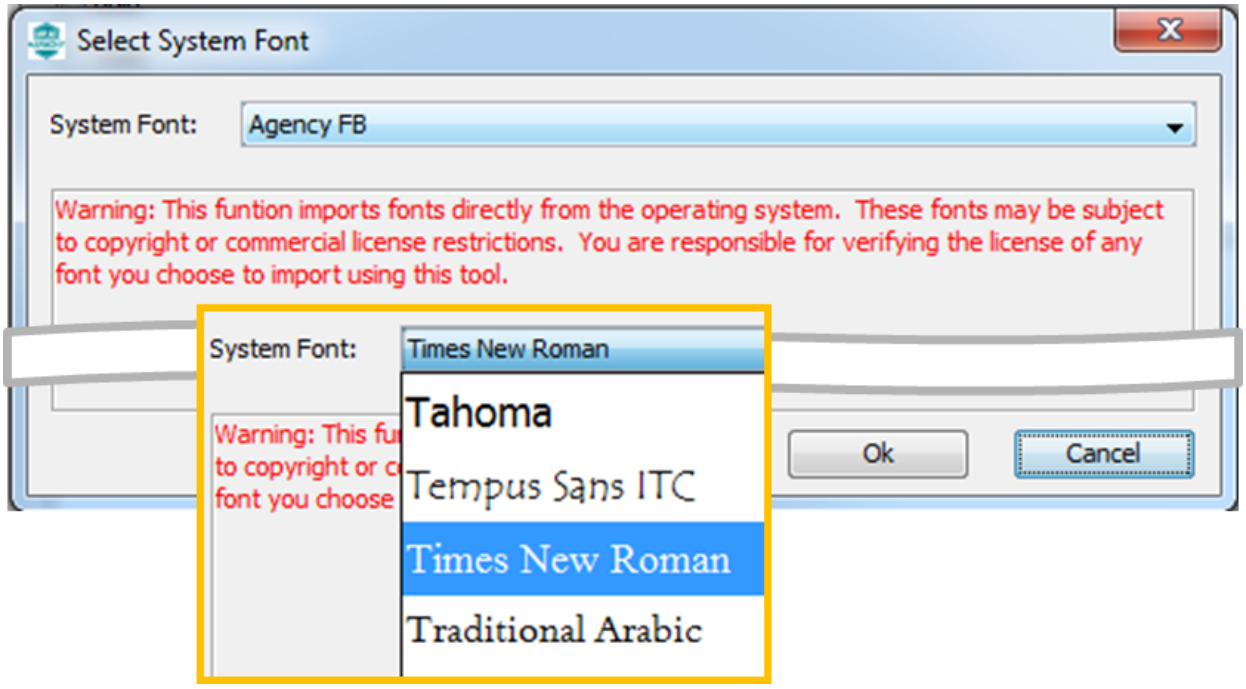
Window Toolbar

There are five icons on the toolbar below the Images tab:

1. **Add Font From File** – Adds a font asset from a file.



2. **Add Installed Font** – Add a font installed on your computer.



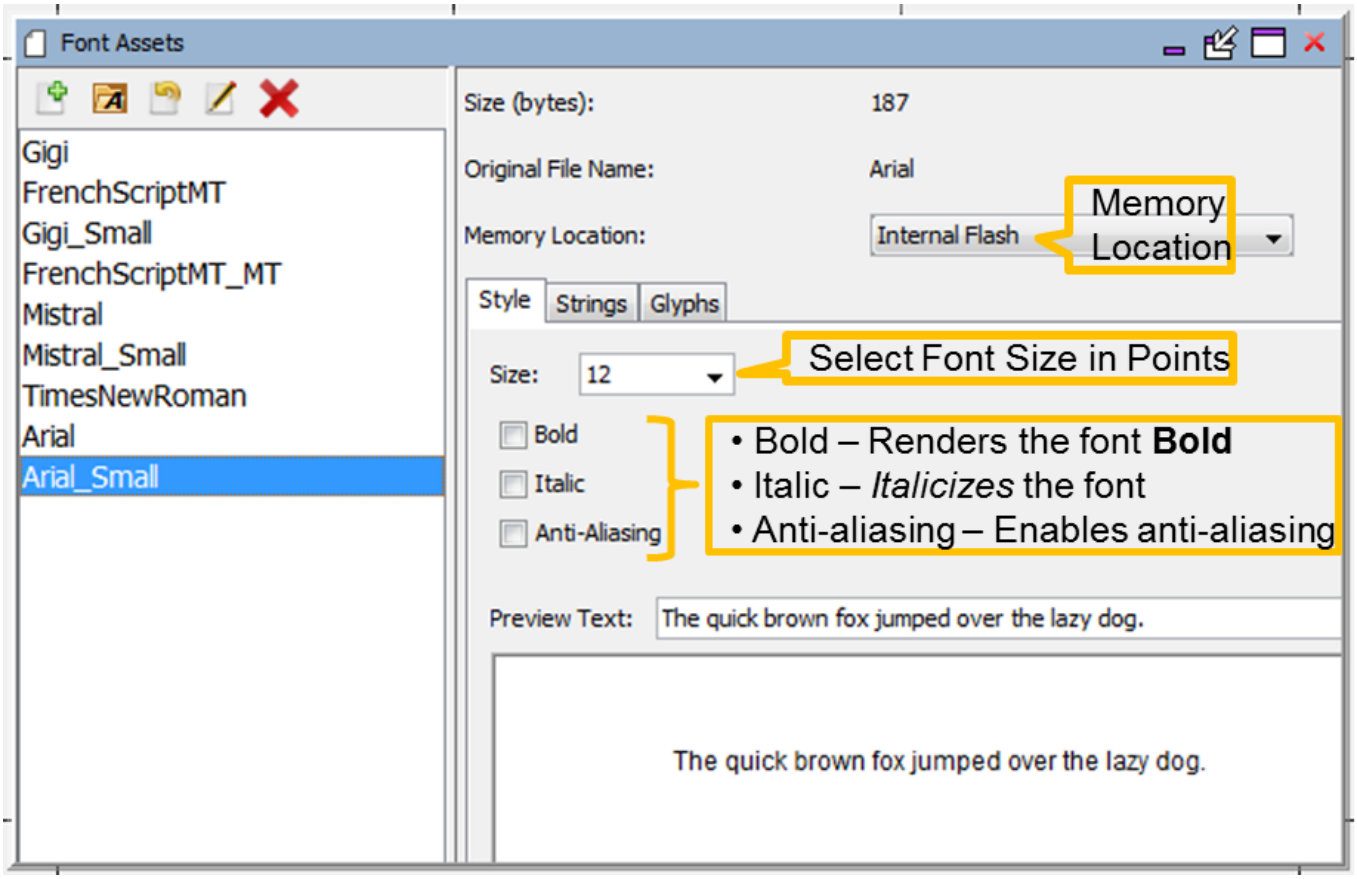
3. **Replace Existing Font Data with New Source Font** – Both Add Font From File and Add Installed Font create a new font asset. This icon allows you to update an existing font asset, importing from a file or using a font installed on your computer.
4. **Rename Selected Font** – Renames an existing font asset. In the example above, the Arial font was installed twice, first as a 16 point font and second as a 12 point font. If added to the fonts assets in this order, the 12 point font will have the name Arial_1. This font asset was renamed to Arial_Small using this tool.

5. **Delete Selected Fonts** – Removes selected font assets from the application.

Sub-tabs

There are three sub-tabs to this window.

Style Sub-tab



The Size (bytes): shown represents the memory needed to store all the font's glyphs. The application only stores the glyphs that are used by static (build-time) strings and by predefined glyph ranges to support dynamic (run-time) strings.

The choices for Memory Location must be defined before the font can be assigned. Go to the Memory Configuration window to add a new location before using it in this sub-tab.

Each font asset consists of a font, size, and some combination of the { Bold, Italic, Anti-Aliasing } options, including selecting none of these options. If you need bold for one set of strings and italic for another, then you will need two font assets, one with Bold checked and a second with Italic checked. The same applies for font sizes. Each font size requires its own font asset. Thus if you need two sizes of Arial, with plain, bold, and italic for each size, you will need 6 separate assets (6 = 2 Sizes x 3).

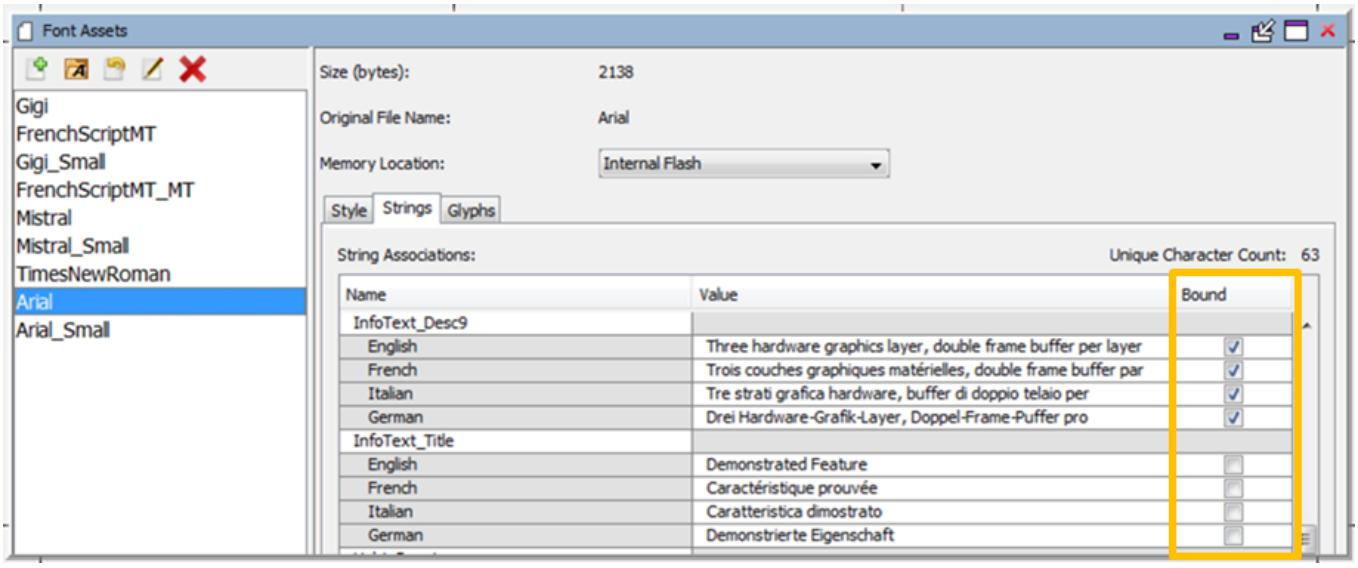
Glyphs are normally (Anti-Aliasing off) stored as a pixel bit array, with each pixel represented by only one bit. Turning on Anti-Aliasing replaces each pixel bit with an 8-bit gray scale, thereby increasing font storage by a factor of 8!

What if a font is chosen that does not support the character types of the text used for a particular language in the application? How can you test and debug this? There a basically two ways:

- Use an external font viewer to examine if the needed glyphs exist
- Configure, build, and run the application and verify the strings are correctly rendered

If the glyphs are not available they will be rendered as rectangles (□).

Strings Sub-tab



The Bound check box accomplishes the same thing as assigning a font to a text string in the Strings Assets window (Window:Strings menu). Assigning a string to a font means that the font will generate glyphs for that string. This is just another way to accomplish the binding of the string text to font.

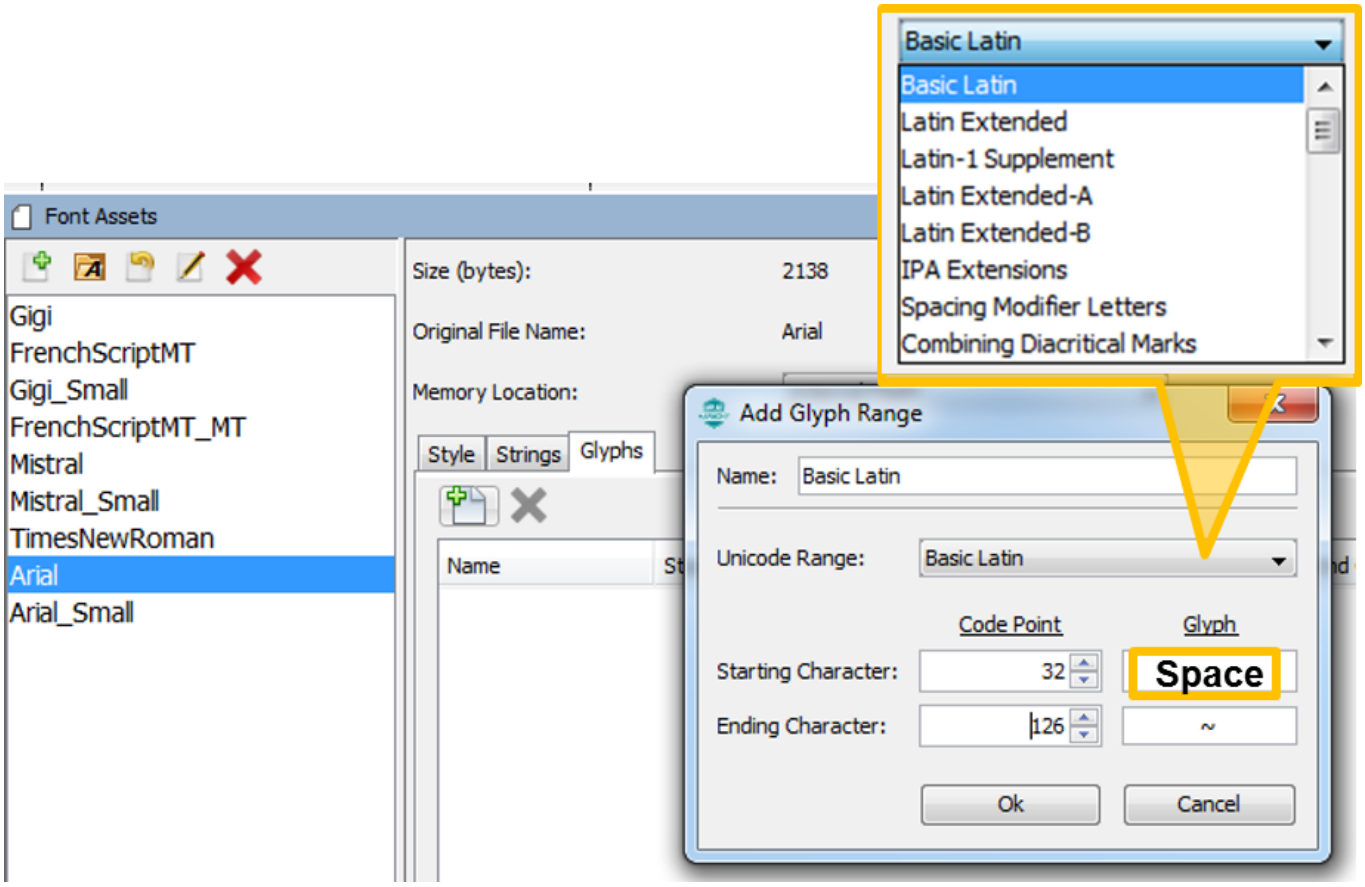
This sub-tab is also useful in a complicated graphics design to see how many strings use a particular font. Lightly-used or unused fonts can be eliminated to free up internal Flash memory.

Glyphs Sub-tab

Note: The word “glyph” comes from the Greek for “carving”, as seen in the word hieroglyph – Greek for “sacred writing”. In modern usage a glyph is an elemental or atomic symbol representing a readable character for purposes of communicating via writing.

The Glyph sub-tab is only used when your application supports dynamic strings. For static (build-time) strings MHGC automatically determines which font glyphs are used based on the characters present in all the strings used by the application’s graphics widgets. Only these glyphs are included as part of the application’s font assets. With dynamic (i.e. run-time) strings this is not possible. This sub-tab allows you to specify which range of glyphs will be used by run-time strings. Once glyph ranges are defined, these glyphs are added to the font glyphs used by static strings.

The Create New Custom Import Range icon ( ) allows you to input a new glyph range for the font. Selecting this icon opens the Font Assets window.



String Table Configuration

Provides information on the String Assets features.

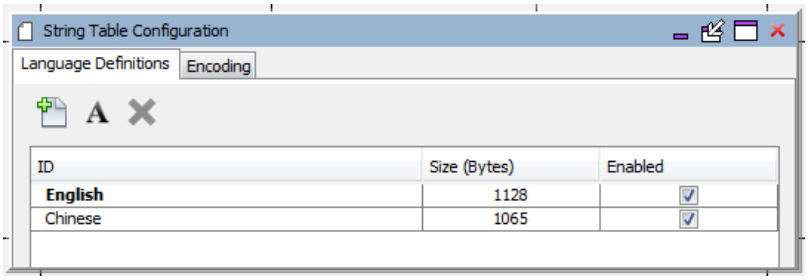
Description

The String Table Configuration window is launched from the Graphics Composer's Asset menu.

Note: There are three dimensions to text support: Languages, Fonts, and Strings. Language "ID" strings are identified when an application supports more than one language. (In the case of single language support, the language default is provided.) Fonts are imported and organized using the Font Assets window. Strings are defined by a string name, and this name is used by widgets to reference the string. For each string and each language supported the glyphs are defined to spell out the string's text and the font is chosen for that text.

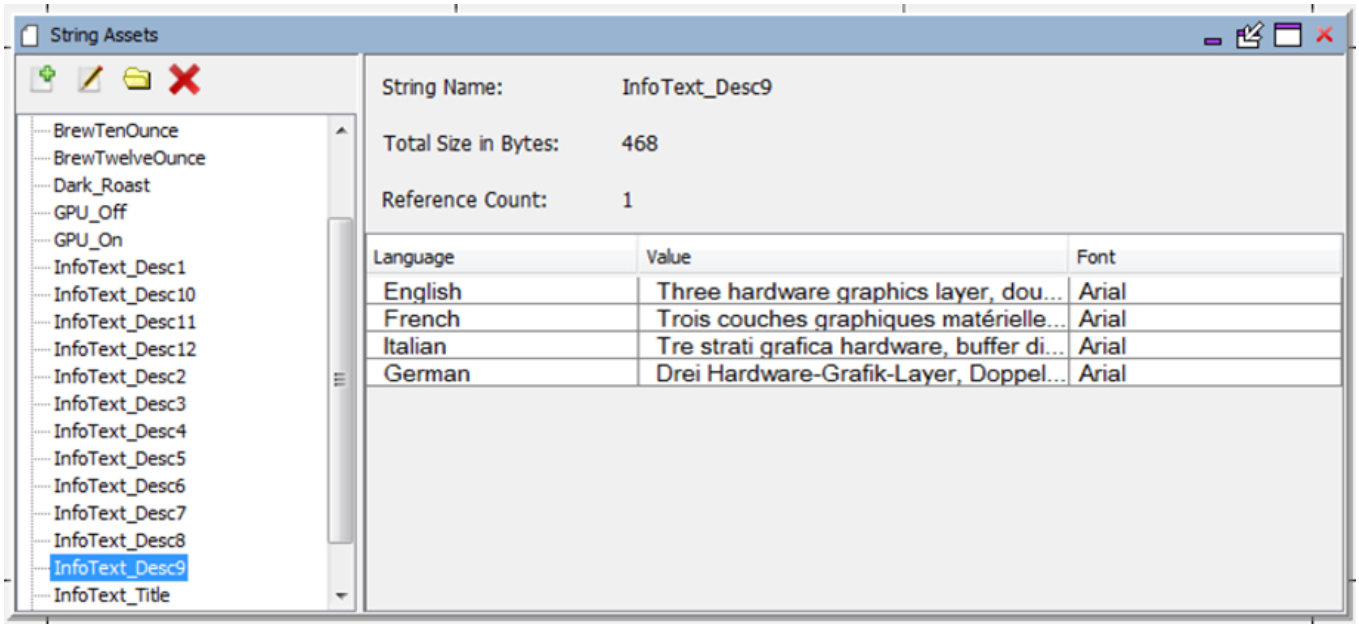
- Languages are managed within the String Table Configuration window (this topic)
- Fonts are managed within the Font Assets window
- Strings are managed within the String Assets window

Within this window, the Languages supported by the application are defined and the encoding for all application glyphs selected.



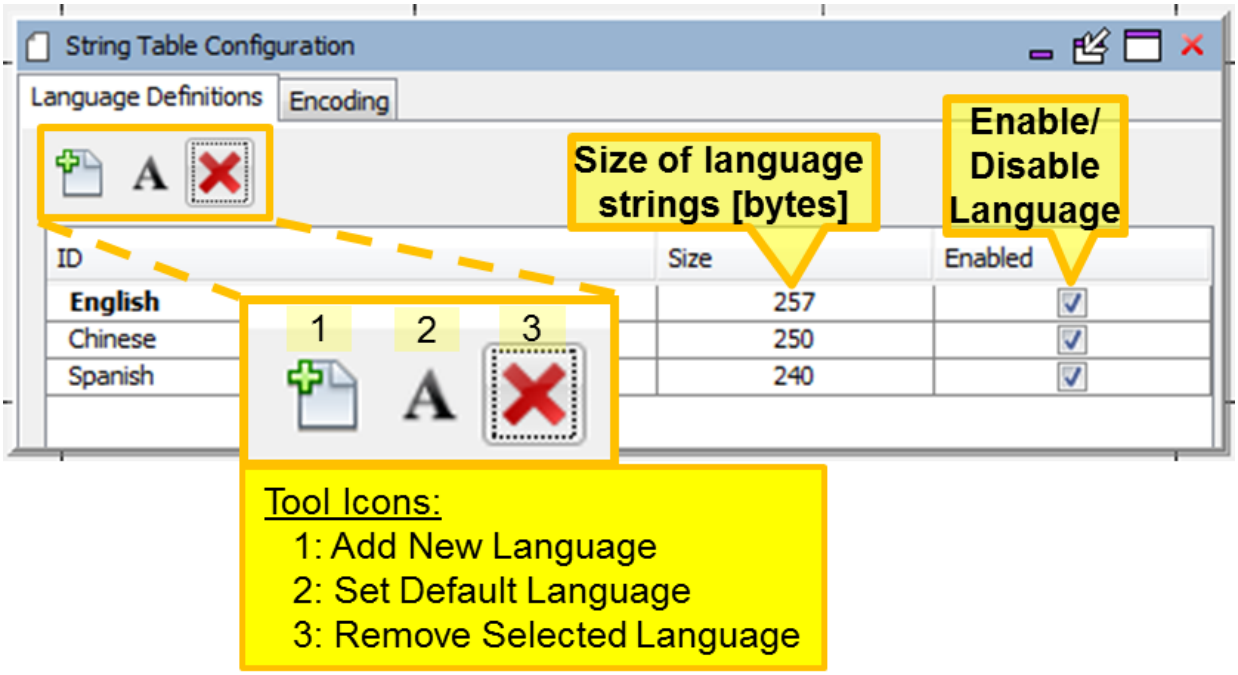
The "ID" string used for each language is merely for ease of use in building the texts to be used. "English", "American", or any other string can be used to identity that language, as long as it is understood by the application's creator when selecting the text to be used for that particular language. Then the application can switch to supporting one of its languages using "ID" strings defined.

Here is an example string asset definition, taken from the Aria Coffee Maker demonstration. This application supports English, French, Italian, and German. The text string "InfoText_Desc9" uses the Arial font, and text for each language is specified within the String Assets window.



Any number of languages can be defined as long as there is memory to store the strings needed.

The following figure shows the String Table Configuration for an application that uses English, Spanish, and Chinese.



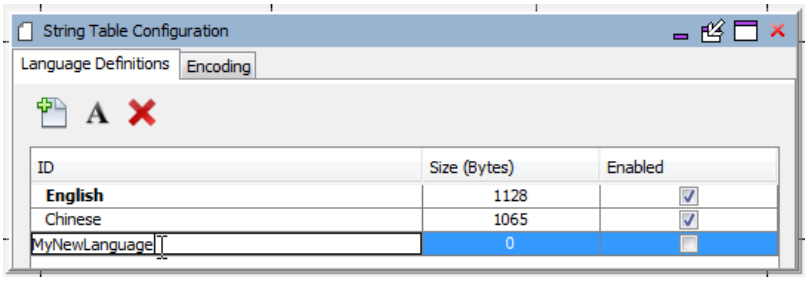
The size of all the strings for each language is shown in the Size column. String size represents the memory allocated for glyph indices for all the strings supporting that language. A language can be enabled/disabled via the check box in the Enabled column. Disabling a language removes it from the application build but keeps it in the project.

Window Toolbar

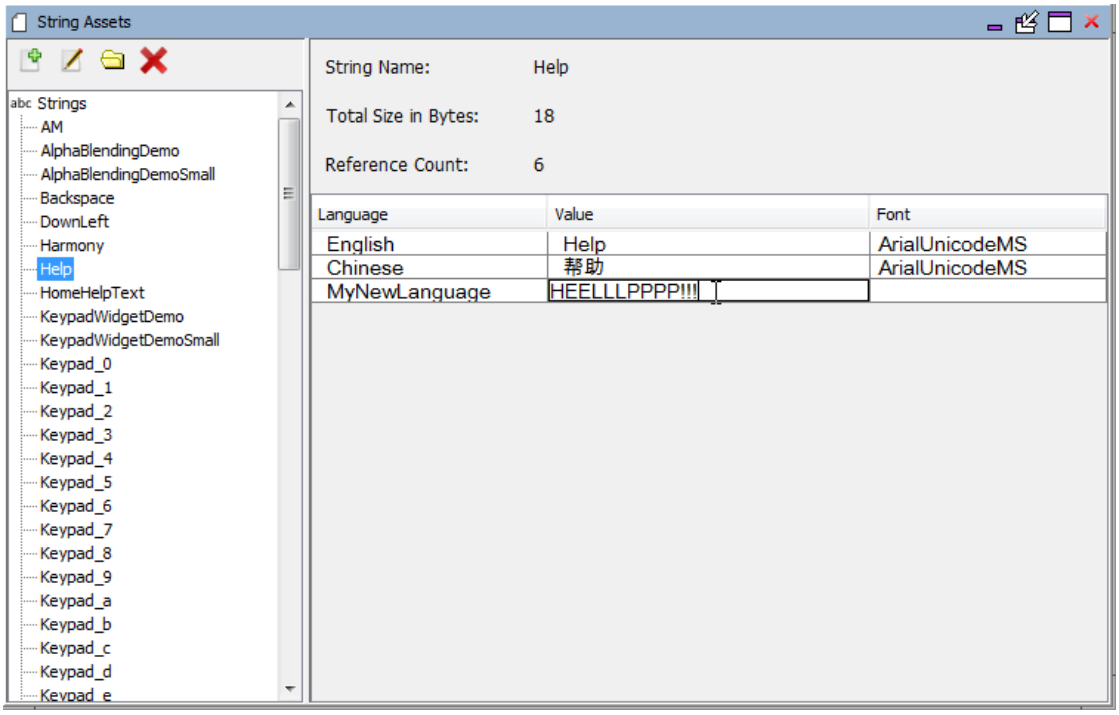
There are three icons on the toolbar:

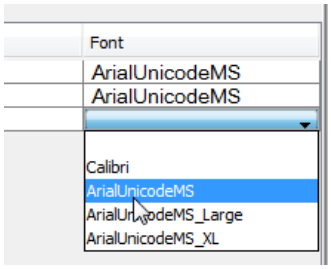
1. **Add New Language** – Adds a new Language.
2. **Set Default Language** – Sets the application's default language. Note, this is different than the abc tool on the Graphics Composer Window toolbar. The abc icon sets the preview language for the Screen Designer panel only. This icon sets the language used by the application after boot-up.
3. **Remove Selected Language** – Removes language from the application.

Clicking **Add New Language** opens a new line, allowing you to select and edit the new language's "ID" string.



Then, for every string defined in the application there will be a line to define the needed text, and to specify the font to be used.





If you don't provide a value for the new language the string will be output as a null (empty string). If you don't provide a Font selection then the string will be output as a series of blocks (?).

The Aria User Interface Library primitive, `LIB_EXPORT void laContext_SetStringLanguage(uint32_t id)`, allows the application to switch between languages using the Language ID `#defines` are specified in the application's `gfx_assets.h` file.

Sub-tabs

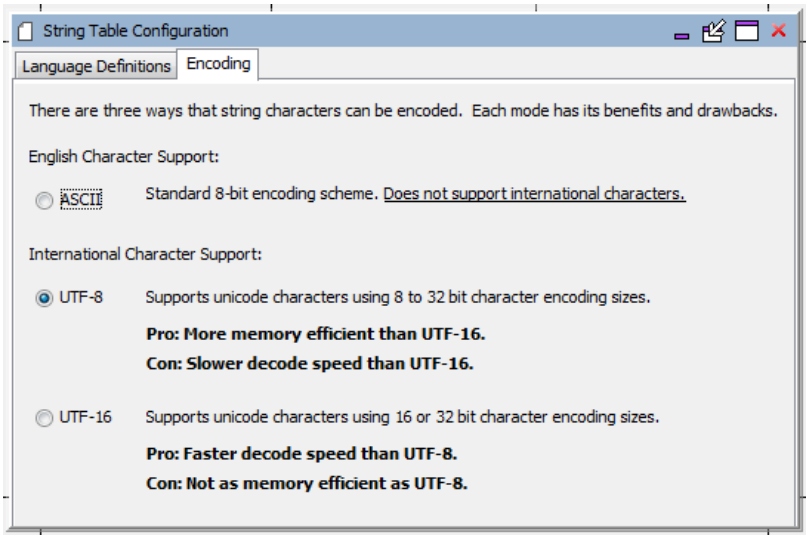
There are two sub-tabs to this window.

Language Definitions Sub-tab

This sub-tab shows the languages defined for the application. A Language can be enabled/disabled to include or exclude it from the application's generation/regeneration under MPLAB Harmony Configurator (MHC). New languages can be added by specifying a text string for the language. With a new language, go to the String Assets window to specify the text and fonts for all defined strings.

Encoding Sub-tab

Selecting the Character Encoding Format Selection Dialog icon gives you three choices for how the characters in all strings in the graphics application are encoded:



The default is ASCII. It is typically the most efficient in terms of memory and processing, but it does not support as many glyphs. Chinese text should be encoded in UTF-8 or UTF-16, but Western language text can be encoded in ASCII to save memory. The trade-off between ASCII, UTF-8, and UTF-16 depends on the application. Changing from UTF-8 to UTF-16 will double the size of all strings in the application. This is because the sizes of all glyph indices double in size. (String sizes are the sizes of glyph reference indices, not the size of the particular font glyphs used to write out the string.)

The memory utilization resulting from an encoding choice can be seen in the Summary sub-tab of the Memory Configuration window.

String Assets

Provides information on the String Assets features.

Description

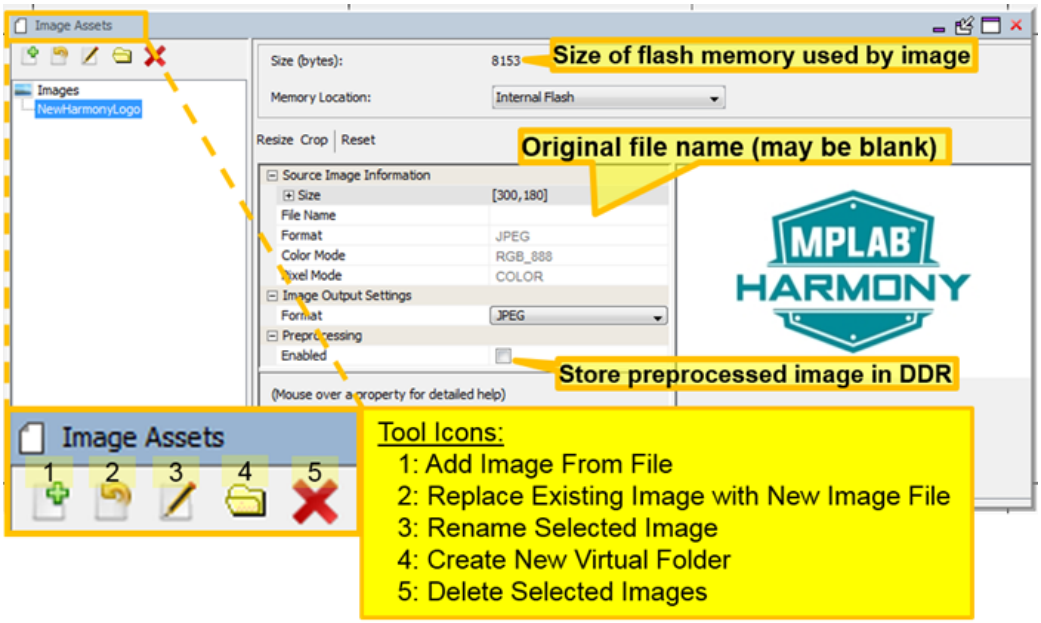
The String Assets window is launched from the Graphics Composer's Asset menu.

The String Assets window supports managing the strings in the application. Strings are referenced by graphic widgets using an application-wide unique name. This unique name is built into an enumeration that the application's C code uses. For each language supported text is defined and a font asset selected.

Note: There are three dimensions to text support: Languages, Fonts, and Strings. Language "ID" strings are identified when an application supports more than one language. (In the case of single language support, the language default is provided.) Fonts are imported and organized using the Font Assets window. Strings are defined by a string name, and this name is used by widgets to reference the string. For each string and each language supported the glyphs are defined to spell out the string's text and the font is chosen for that text.

- Languages are managed within the String Table Configuration window
- Fonts are managed within the Font Assets window
- Strings are managed within the String Assets window (this topic)

The following figure shows an example taken from the Aria Coffee Maker demonstration. The string name, InfoText_Desc9, defines a string asset that is used by the application.



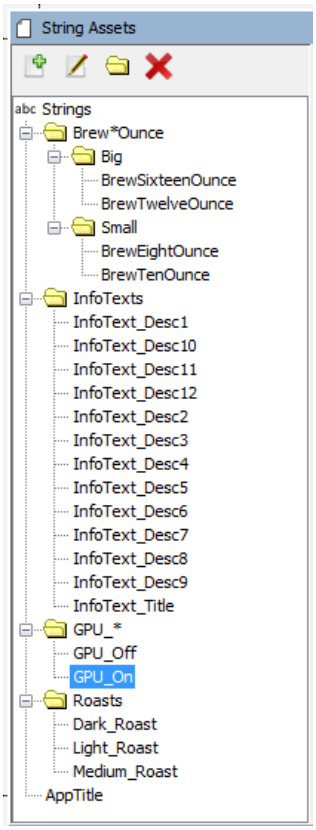
The Total Size in Byte: for a string asset represents the memory needed to store the glyph indices for all the text defined for that string asset. Adding more text will increase the number of glyph indices needed thus increasing the size of the string's memory. Adding another language will do the same, since the number of glyph indices also increases. Changing the font does not increase the size of the string's memory, but may increase the size of the font chosen if it is a "bigger" font and adds more glyphs to the new font. (By "bigger" we mean a font with more pixels, for example because it is bigger in size, or perhaps because it is anti-aliased and the original font was not.)

Note: The Reference Count shown reflects the number of build-time references to the string. Dynamic uses of a string, such as through macros or events, is not reflected in this number.

Window Toolbar

There are four icons on the toolbar:

1. **Add New String** – Adds a new string.
2. **Rename Selected Item** – Allows renaming the string.
3. **Describe Selected String** - Provides a *Description* field value for selected string.
4. **Create New Virtual Folder** – Creates a new virtual folder, allowing you to organize strings in a hierarchy. Here's an example reorganization of the existing strings. Note the order of virtual folders or items in the list is strictly alphabetical. Virtual folders and string asset organization is merely for the convenience of the developer. Neither has an effect on how the application is built.

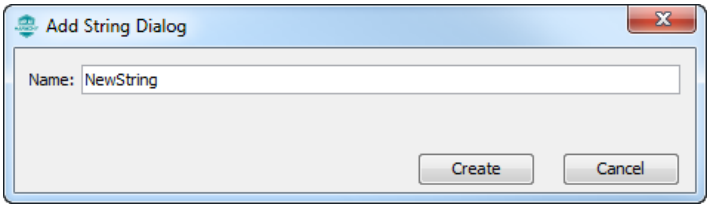


5. **Delete Selected Items** – Deletes selected strings from the application.
6. **Import String Table** - Imports an Excel CSV (Comma Separated Value) file to replace the current string table.
7. **Export String Table** - Exports the current string table as an Excel CSV (Comma Separated Value) text file.

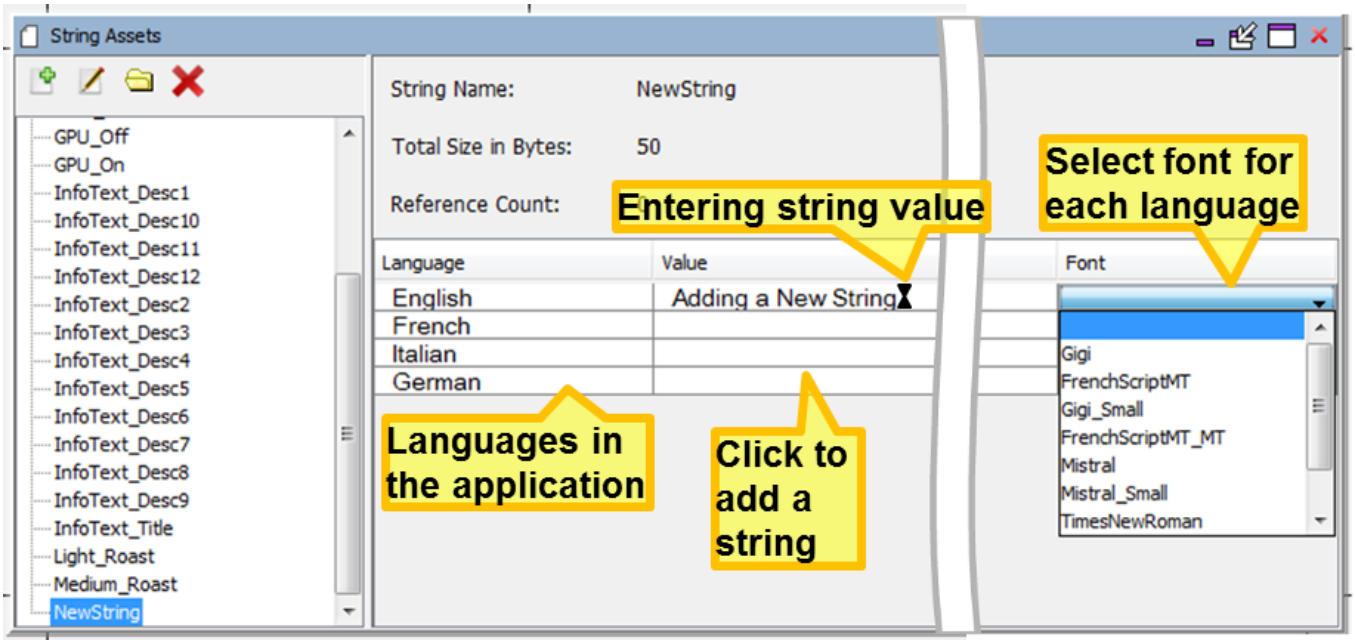
Creating New Strings

To create a new string, click Add New String ( ).

Selecting this icon opens the Add String dialog to name the string. The text chosen for the string name should be acceptable as a C variable.



After entering the new string's name and click Create, the following String Assets window appears.



In the String Assets window, there will be a line for each of the languages defined for the application. Provide the string text and font for each of the languages. An empty string will be used if the text is not provided. Not providing a font causes the string to be rendered as a string of boxes (□).

Importing and Exporting String Tables

Importing an Excel CSV (Comma Separated Values) file replaces the existing string assets table. Exporting creates an Excel CSV file that can be imported into another project or target configuration. Exported string tables can be manipulated in Excel, even combining multiple string tables into a single string table that can then be imported.

If the string asset table contains UTF-8 then the file cannot be directly loaded into Excel. Instead, within Excel create a new sheet. Import the string table using *Get Data*, selecting *From File, From Text*, or *CSV*. Then in the dialog window change the *File Origin* to *Unicode (UTF-8)*.

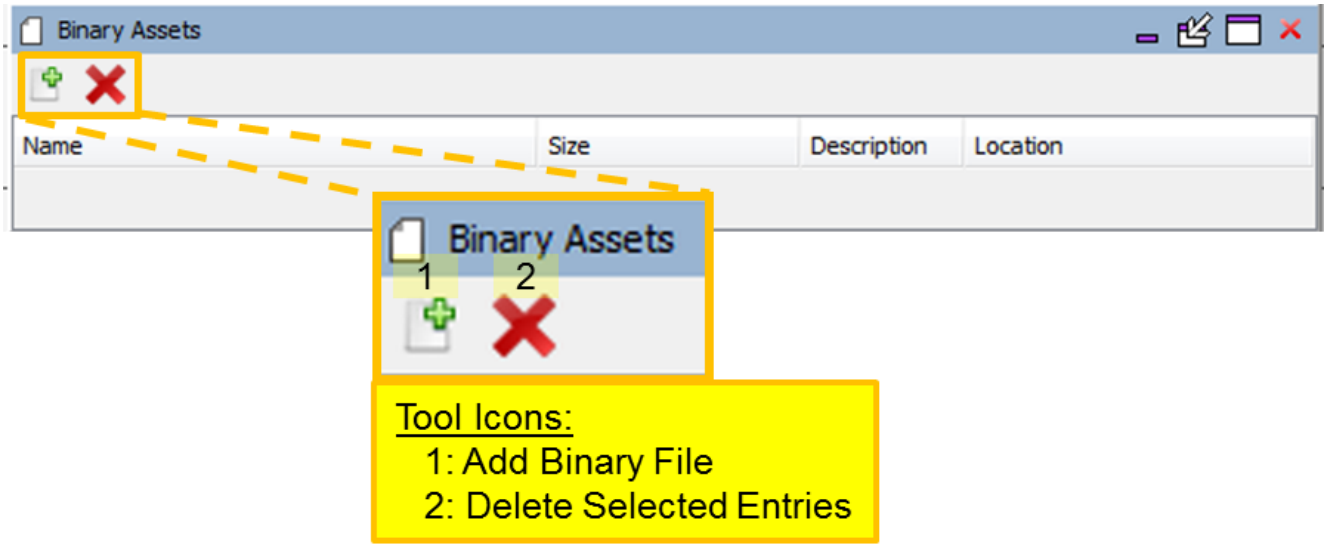
Note: Excel does not support importing UTF-16.

Binary Assets

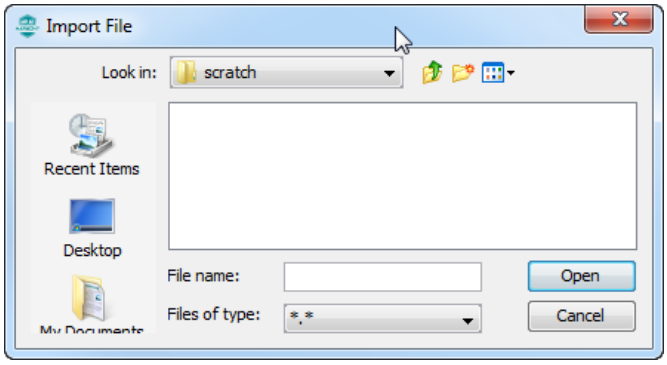
Provides information on the Binary Assets features.

Description

The Binary Assets window is launched from the Graphics Composer's Asset menu.



Selecting the Add Binary File icon (opens the Import File dialog.



This supports any formatted binary file. Developers can then add a custom-coded decoder to support the format implied by the imported file. (A future version of the GFX library will include a bin2code utility in support of this feature.)

MHGC Tools

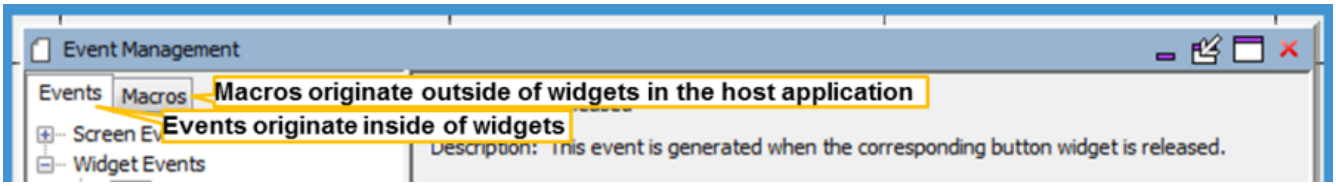
The Tools menu supports managing all graphics events, using a global palette, and estimating heap memory usage.

Event Manager

This section provide information on the Event Manager.

Description

The Graphics Composer Event Manager provides a GUI interface to manage all of the events associated with a graphics application. In a general sense, an event is an action or occurrence that is processed by software using an “event handler”. Button pushes or keystrokes are widely recognized and handled events. Events related to a touch screen are commonly called “gestures”. This GUI allows the assignment of actions to events associated with graphics widgets and to events outside of the graphics library. Under the Graphics Composer Event Manager tab there are two sub-tabs, one for “Events” and a second for “Macros”.



The following table summarizes the difference between "events" and "macros" and provides examples of each instance of source to destination:

Differences Between Events and Macros

Source	Inside of Graphics (Destination)	Outside of Graphics (Destination)
Inside of Graphics	"Event" Example: Button changes button text	"Event" Example: Button changes MEB2 LED color
Outside of Graphics	"Macro" Example: Mounting SD card changes screen	Not supported by Event Manager Tool

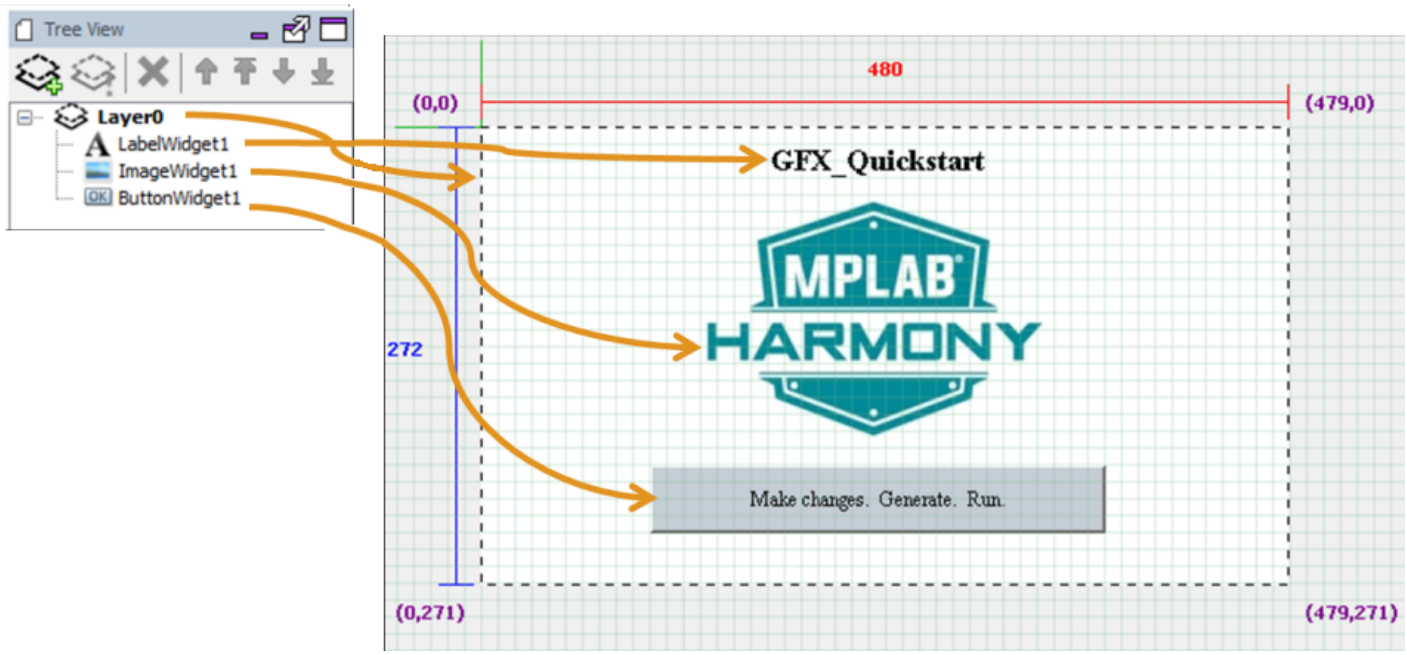
“Events” under the first tab are generated from within graphics widgets and can manipulate the properties of screen widgets or set semaphores that engage with the rest of the application. “Macros” are executed outside of graphics widgets by other parts of the application. “Macros” allow the application to change widget properties or behavior.

Both “Events” and “Macros” event handlers can be built using collections of “Template” actions or using “Custom” developer-provided code. Most widget properties have an associated Template action that can be used to manipulate that property in an event handler (either “Event” or “Macro”). For more information on properties and related actions, see the discussion on the Properties Window below.

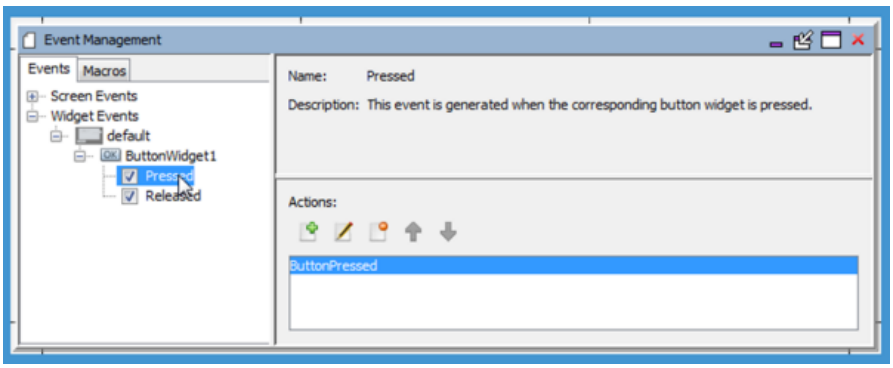
To explore these capabilities, let’s look at the Aria Quickstart project after the completion of the Adding an Event to the Aria Quickstart Demonstration Quick Start Guide.

Graphics Composer Events

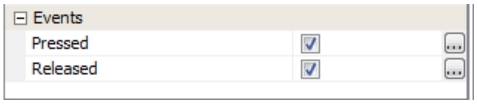
The Graphics Composer Screen Designer shows that there is one layer and three widgets in this demonstration.



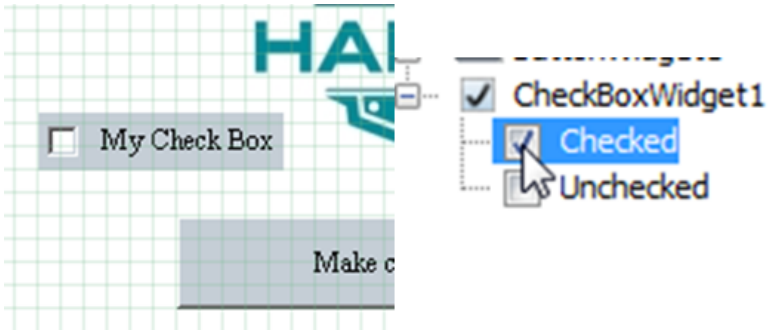
Of the three widgets shown above, only ButtonWidget1 can have events associated with it, one for button pressed and a second for button released. This can be seen in the Graphics Composer Event Manager window, which is available from the Tools menu:



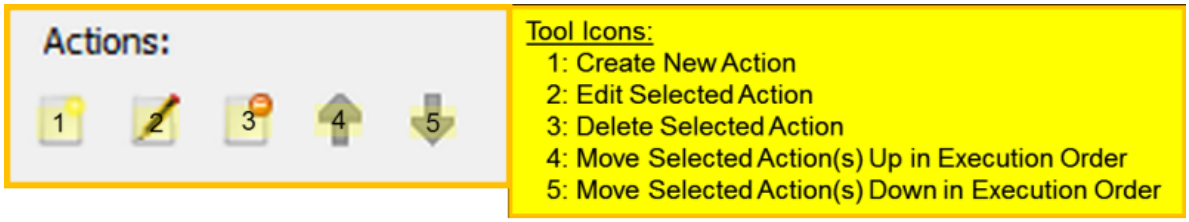
The events shown under “ButtonWidget1” are mirrored in the widget’s properties. Selecting or clearing an event in one window does the same in the other window, thus enabling (selecting) or disabling (clearing) the corresponding event.



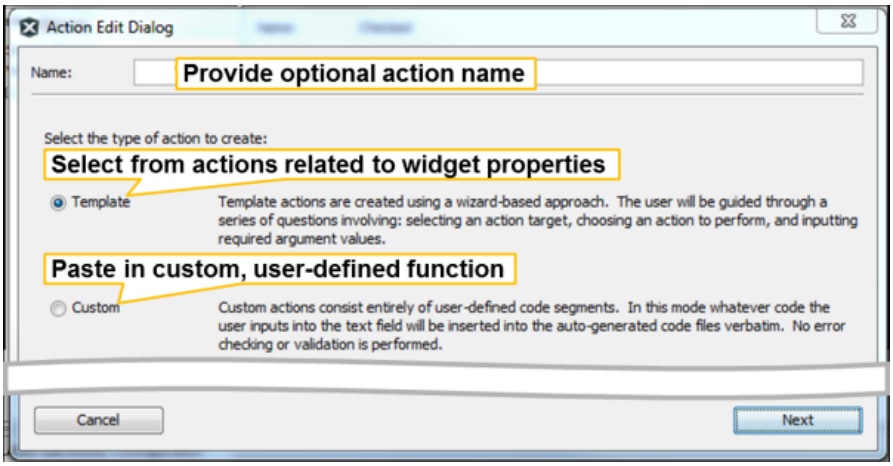
We can add a Check Box widget to the applications display and then use the Event Manager to assign actions to the widget’s events. A Check Box widget has two events, one for being “Checked” (i.e., selected) and another for being “Unchecked” (i.e., cleared). Enabling the “Checked” event then allows the selection of the action or actions for that event.



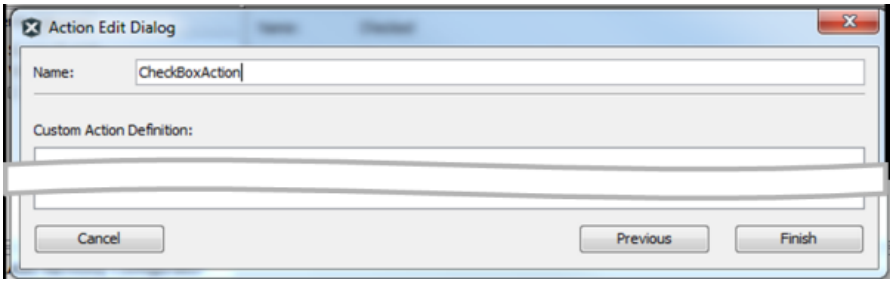
The Actions: sub-window has five tool icons for managing the actions associated with an event:



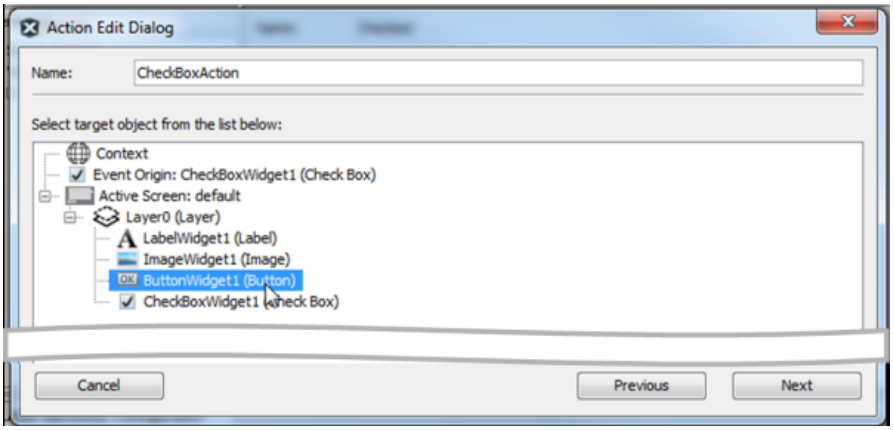
Clicking the Create New Action icon () opens the Action Edit dialog.



If you select Custom and click **Next**, you will see the following dialog. Unfortunately, there is no C code error checking with this window. It just copies the code into `libaria.c` and `libaria.h`. If there is a problem with the code you will not know about it until you try to build your application. An alternative is just to type a comment such as `/*My event goes here*/`, generate the code, and then find out where this comment landed in the code. (Typically, inside `libaria_events.c`, or `libaria_macros.c`) You can then write the action routine from within the MPLAB X IDE editor and compile just that file to debug the code written.

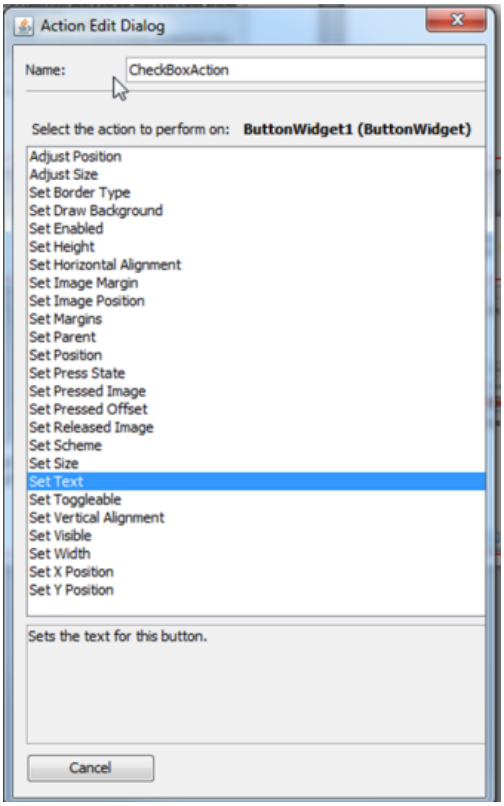


If you select Template, the Action Edit dialog will update, as follows. Select ButtonWidget1.

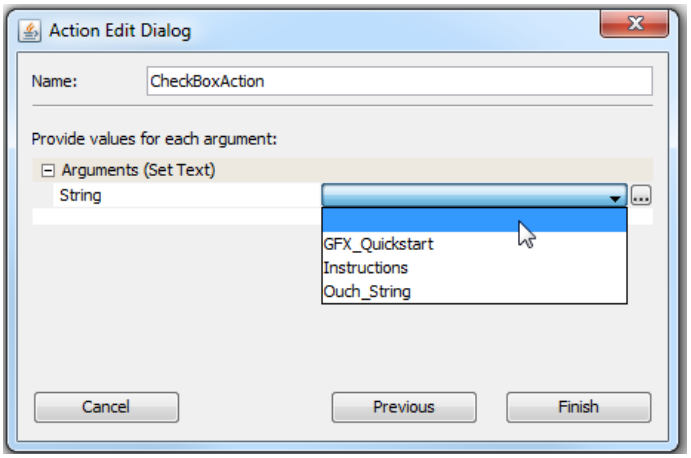


As shown previously, you next need to select the widget that you want to manipulate with this action. Note that the event originated with CheckBoxWidget1, but the event's action can manipulate any of the existing widgets. In this case, ButtonWidget1 has been selected. Clicking Next

will then bring up a list of the actions available in manipulating a button widget.



You can select the “Set Text” action, which will then change the button’s text property, followed by NEXT, which will open a dialog to select the text string for this action.



You can then select from the available (already defined) strings which text to use for the button’s text field. Press the Finish button to complete the definition of this action.

Screen Events

As shown previously, the Graphics Composer Event Manager, Events sub-tab supports screen events when the screen is visible (On Show) and hidden (On Hide). These events can define event handlers based on Template actions or Custom, user-defined code.

Widget Events

Not all widgets can generate an event. For example, a Label Widget has nothing to generate, it just sits there on the screen, labeling. Here is a list of the widgets that can generate an event:

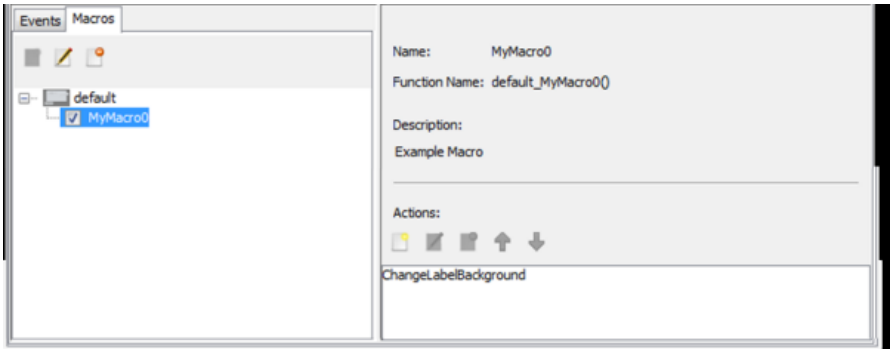
- Button – Pressed and Released events
- Check Box – Checked and Unchecked events
- Draw Surface – Draw Notification event
- Image Sequence – Image Changed event

- Key Pad – Key Click event
- List Wheel – Select Item Changed event
- List – Selection Changed event
- Progress Bar – Value Changed event
- Radio Button – Selected and Deselected event
- Scroll Bar – Value Changed event
- Slider Widget – Value Changed event
- Text Field – Text Changed event
- Touch Test – Point Added event

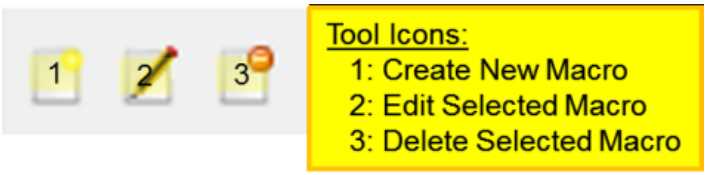
Graphics Composer Macros

Macros implement event handlers for events that originate outside of graphics primitives such as widgets and are designed to change or manipulate widgets inside of the graphics part of an application. (Events that originate outside of graphics and don't touch the graphics part of the application are outside of the scope of the Graphics Event Manager and are not discussed here.)

The following figure shows a simple example of a macro.

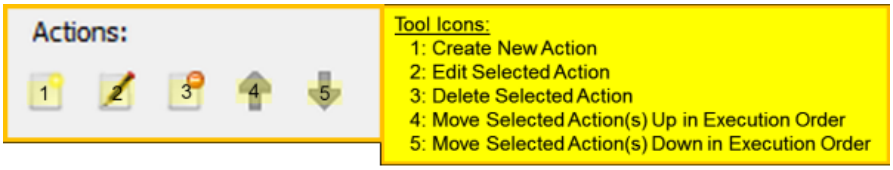


The toolbar for Macros has three icons.

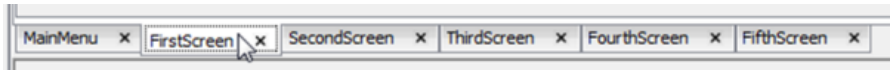


Creating a new macro and selecting its actions is just like that of a widget event:

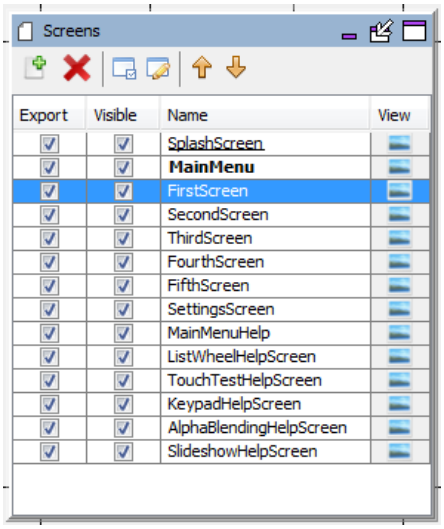
1. Create a new macro using the "Create New Macro" tool. The check box to the left of the new macro's name enables/disables the macro. Clearing it removes the macro from the next code generation.
2. Select the new macro and edit it using the second icon (shown previously).
3. In the Actions: window, select Create New Action. An optional name can be provided in the Name: box. You can then choose to use a Template and select a predefined action or Custom to create a customized action.



4. If you chose a "Custom" action, proceed as discussed previous in Graphics Composer Events. When using templates the next step is to choose the target widget for the action. This choice is limited to those only the widgets in the currently "active" screen. If your application has multiple screens and the widget you are targeting is not part of the currently active screen you need to change the active screen.
   - Changing the active screen can be done by selecting the corresponding screen tab at the bottom of the Graphics Composer Screen Designer



- Alternately, you can switch using the Graphics Composer Manager:Screens tab



5. After selecting the target widget for this macro, click Next button to select an action related to this widget. (Just as with template-based widget events.) The macro can contain more than one action, targeting more than one widget.

Graphics Events Test Bed

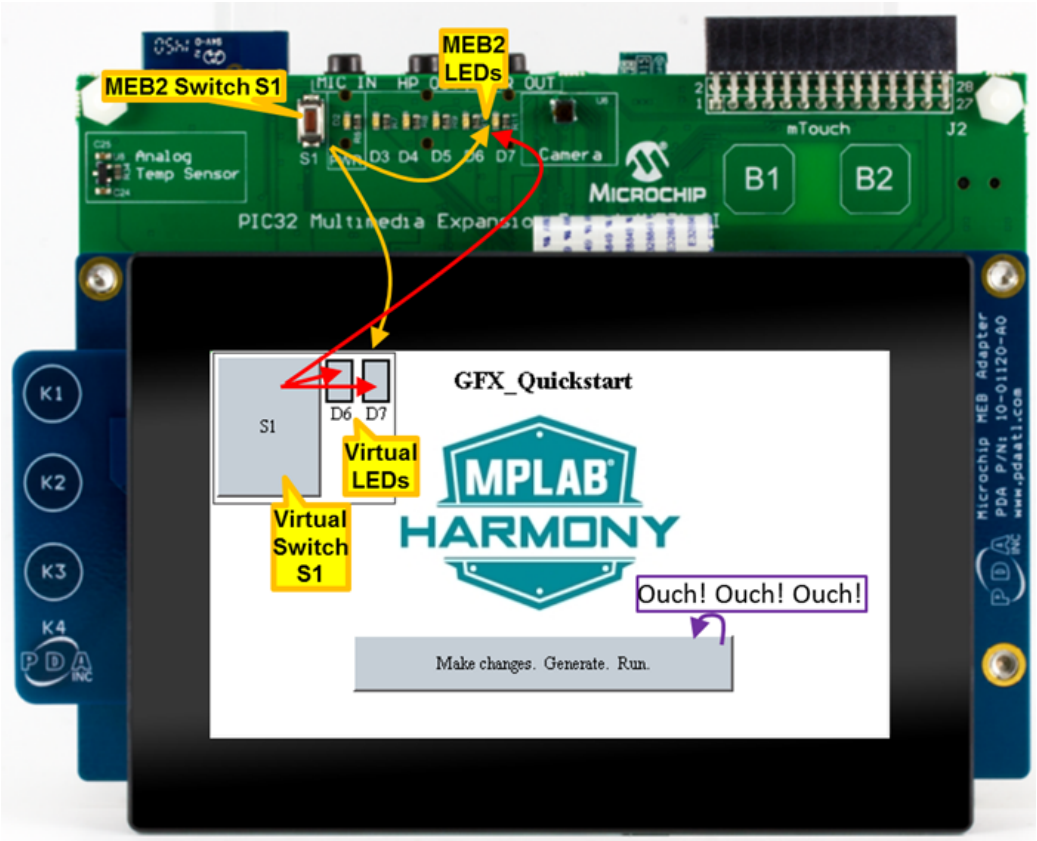
Additional examples of events and macros can be found in the MPLAB Harmony project found in *./apps/examples/events_testbed*. This project is based on the Quick Start Guide “Adding an Event to the Aria Quickstart Demonstration” found in Volume 1 of MPLAB Harmony’s built-in documentation.

This project has target configurations for PIC32MZ DA and EF starter kits with the MEB2 graphics board. It demonstrates the following events/macros:

Event Testbed

Source	Inside of Graphics (Destination)	Outside of Graphics (Destination)
Inside of Graphics	"Event" Button changes button text from "Make Changes. Generate. Run" to "Ouch! Ouch! Ouch!"	"Event" Virtual Switch S1 changes MED2 LEDs D6 and D7 on/off via boolean semaphore
Outside of Graphics	"Macro" APP_Tasks changes color scheme for Virtual LEDs D6 and D7 between LED_OFF and LED_ON	Not supported by Event Manager Tool MEB2 S1 changes MEB2 LEDs D6 and D7

Asserting the “Make Changes. Generate. Run” button on the display changes its text to “Ouch! Ouch! Ouch!”. Pressing the MEB2’s Switch S1 changes the LED D6 and D7 on the MEB2 board as well as changing the virtual LEDs D6 and D7 on the display. Pressing the display’s virtual S1 switch does the same.



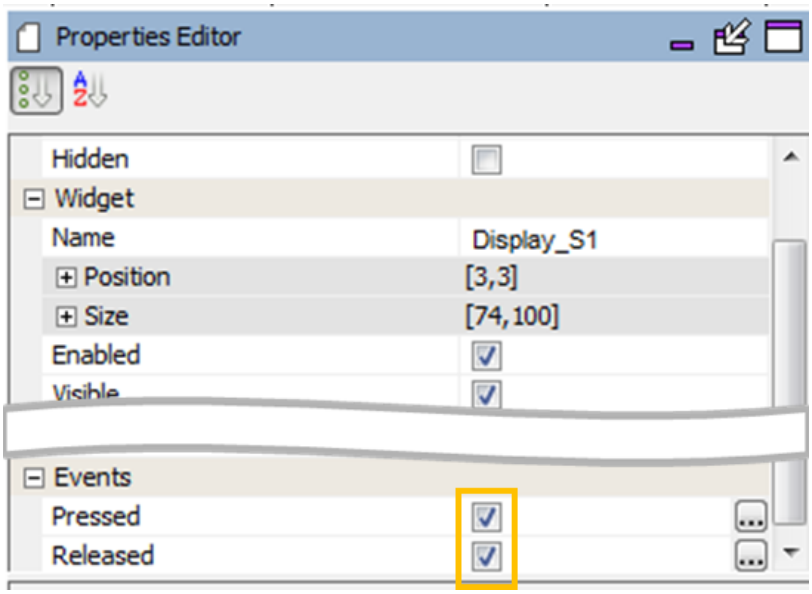
The application's events are defined in `libaria_events.c`:

```
#include "gfx/libaria/libaria_events.h"

// CUSTOM CODE - DO NOT DELETE
extern bool bDisplay_S1State;
// END OF CUSTOM CODE

// ButtonWidget1 - PressedEvent
void ButtonWidget1_PressedEvent(laButtonWidget* btn)
{
// ButtonDown - Set Text - ButtonWidget1
laButtonWidget_SetText((laButtonWidget*)ButtonWidget1,
laString_CreateFromID(string_OuchOuchOuch));
}

// ButtonWidget1 - ReleasedEvent
void ButtonWidget1_ReleasedEvent(laButtonWidget* btn)
{
// ButtonUp - Set Text - ButtonWidget1
laButtonWidget_SetText((laButtonWidget*)ButtonWidget1,
laString_CreateFromID(string_Instructions));
}

// Display_S1 - PressedEvent
void Display_S1_PressedEvent(laButtonWidget* btn)
{
// CUSTOM CODE - DO NOT DELETE
bDisplay_S1State = true;
// END OF CUSTOM CODE
}
```

```
// Display_S1 - ReleasedEvent
void Display_S1_ReleasedEvent(laButtonWidget* btn)
{
// CUSTOM CODE - DO NOT DELETE
bDisplay_S1State = false;
// END OF CUSTOM CODE
}
```

The *ButtonWidget1* changes the text using the `laButtonWidget_SetText` function. Details on how this is accomplished are discussed in the Quick Start Guide “Adding an Event to the Aria Quickstart Demonstration”.

The *Display_S1* widget just sets a Boolean semaphore `bDisplay_S1State`. Creating the events for the Display_S1 virtual switch is easy, just enable the widget’s events in the widget’s properties:



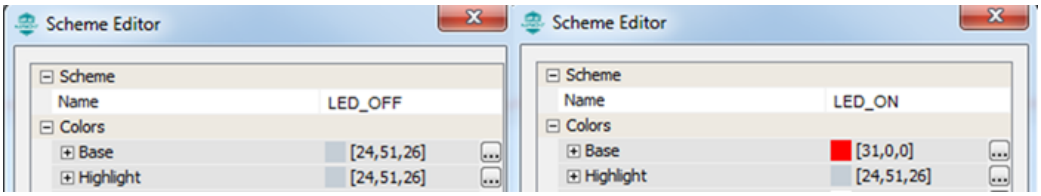
This will create empty event handlers in `libaria_events.c`, which can then be modified to change the boolean semaphore `bDisplay_S1State` as shown above.

The application’s macros are defined in `libaria_macros.c` change the coloring scheme for the display’s virtual LEDs:

```
#include "gfx/libaria/libaria_macros.h"

void LEDsTurnOn(void)
{
if(laContext_GetActiveScreenIndex() != default_ID)
return;

// TurnOnDisplayD6 - Set Scheme - MEB2_LED_D6
laWidget_SetScheme((laWidget*)MEB2_LED_D6, &LED_ON);
// TurnOnDisplayD7 - Set Scheme - MEB2_LED_D7
laWidget_SetScheme((laWidget*)MEB2_LED_D7, &LED_ON);
}


void LEDsTurnOff(void)
{
if(laContext_GetActiveScreenIndex() != default_ID)
return;

// TurnOffDisplayD6 - Set Scheme - MEB2_LED_D6
laWidget_SetScheme((laWidget*)MEB2_LED_D6, &LED_OFF);
// TurnOffDisplayD7 - Set Scheme - MEB2_LED_D7
laWidget_SetScheme((laWidget*)MEB2_LED_D7, &LED_OFF);
```

```
}
```

The difference between the color scheme LED_OFF and LED_ON is only in the base color:



The macros *LEDsTurnOn* and *LEDsTurnOff* are called from the application's main task loop, `APP_Tasks`. The work of controlling the LEDs is done in the APP_STATE_SERVICE_TASKS case.:

```
#include "gfx/libaria/libaria_macros.h"
bool bMEB2_S1State = false;
bool bDisplay_S1State = false;
bool bLED_State = false;
bool bLED_StateNow;

void APP_Tasks ( void )
{

/* Check the application's current state. */
switch ( appData.state )
{
/* Application's initial state. */
case APP_STATE_INIT:
{
bool appInitialized = true;

if (appInitialized)
{
appData.state = APP_STATE_SERVICE_TASKS;
}
break;
}

case APP_STATE_SERVICE_TASKS:
{
bMEB2_S1State = !BSP_SWITCH_S1StateGet(); // Closed --> grounded

bLED_StateNow = bMEB2_S1State || bDisplay_S1State;
if ( bLED_State != bLED_StateNow )
{// LED state has changed
if ( bLED_StateNow )
{
BSP_LED_D6On(); // MEB2 LED D6 On
BSP_LED_D7On(); // MEB2 LED D7 On
LEDsTurnOn(); // Turn display LEDs on
}
else
{
BSP_LED_D6Off(); // MEB2 LED D6 Off
BSP_LED_D7Off(); // MEB2 LED D7 Off
LEDsTurnOff(); // Turn display LEDs off
}//end if ( bMEB2_S1State || bDisplay_S1State )
bLED_State = bLED_StateNow; // Remember new state
}
break;
```

```
}

/* TODO: implement your application state machine.*/

/* The default state should never be executed. */
default:
{
/* TODO: Handle error in application's state machine. */
break;
}
}
}
```

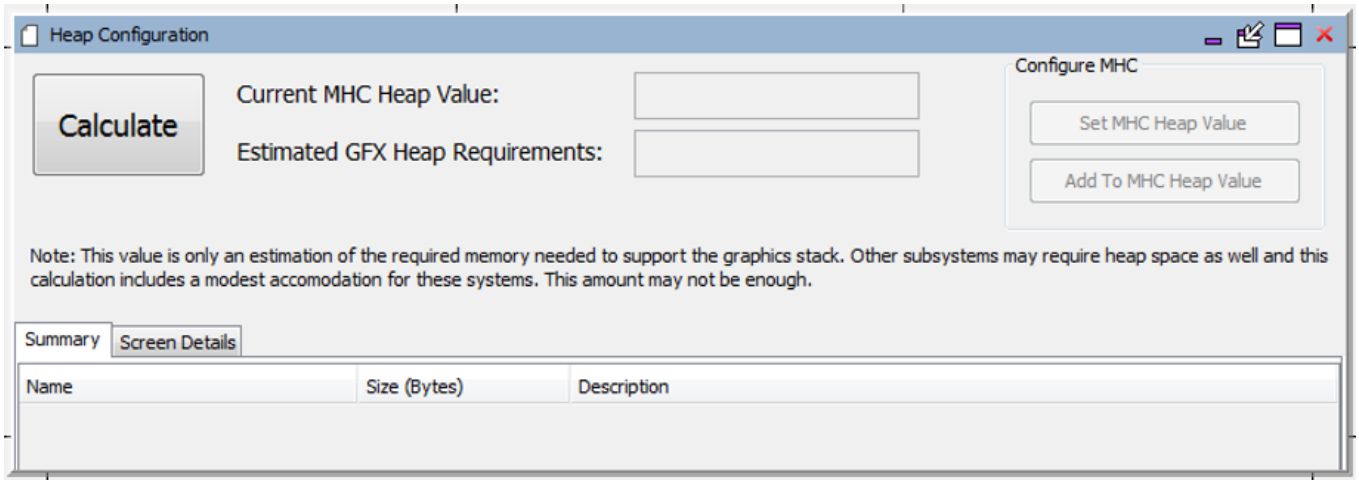
Heap Estimator

Provides information on heap space allocation.

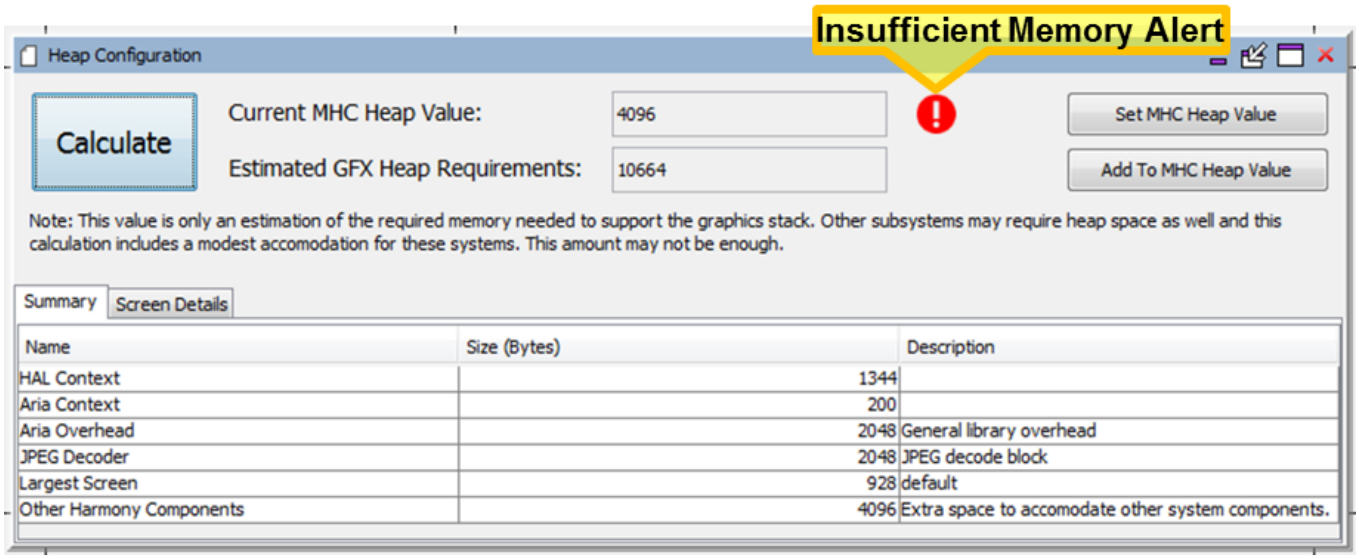
Description

Many parts of a graphics design are implemented using memory allocated from the application's heap space. Therefore, it is important to allocate sufficient memory for the heap. This tool can estimate heap usage by the allocation based on the widgets, layers, screens, and decoders currently in the design.

When launching the tool from the Tools menu, the Heap Configuration window appears.



Clicking **Calculate** estimates heap usage. The following figure shows what occurs within the Aria Quickstart demonstration if the heap space is only 4096 bytes:



The Summary tab shows how the estimated heap requirements was derived by summing up all the sizes shown under the "Size (Bytes)" column.

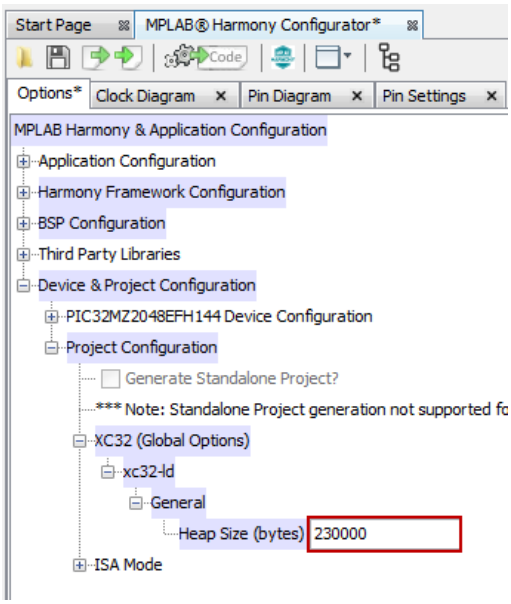
Note that the largest contribution comes from the screen requiring the largest heap allocation (in this case MainMenu).

If there is insufficient memory allocated to the heap, an exclamation point (!) appears in the window. If you hold your mouse pointer over this icon, the following message appears:

> The current MHC heap value is not high enough to accommodate the estimated requirements of the graphics stack. Please adjust the value.

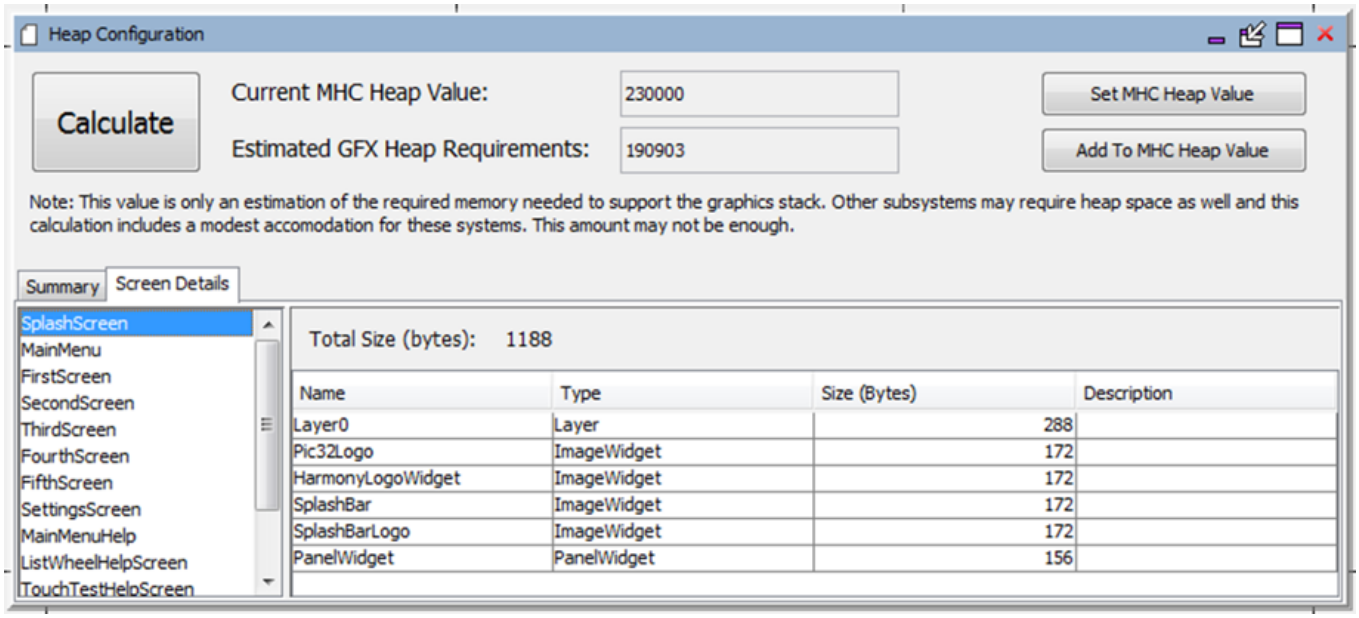
You can click **Set MHC Heap Value** to reset the heap allocation to match the estimated requirements. Selecting **Add to MHC Heap Value** adds the estimated heap requirements to the current heap value. (In the case above, this would change the heap allocation to 4096+10664 bytes.)

Alternately, you can set the heap allocation to a larger value by going to the MPLAB Harmony Configurator window, selecting the Options tab and setting the Heap Size within *Device & Project Configuration* > *Project Configuration*.



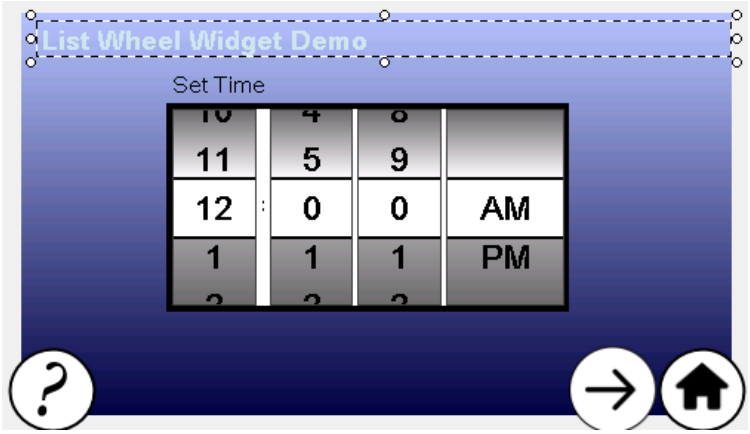
The Screen Details tab (from the Aria Showcase demonstration) shows screen-by-screen the heap space needed for each layer and widget on the screen selected.

Note: After you have updated the Heap Size, either using the Heap Estimator tool or by directly editing the value as shown above, you must regenerate the project using the Generate Code button. This will update the actual heap size value used in building the application.



Clicking the “Name” column will alphabetize the list. Clicking the “Size (Bytes)” column sorts the assets by size, with the largest at the top and smallest at the bottom.

This sub-tab can help in managing the application’s utilization of heap space. For example, excess use of cached backgrounds for widgets can become ruinously expensive, expanding the application’s need for heap well beyond the capabilities of the device. As an example, consider a screen label from the Aria Showcase demonstration.



The Heap Estimator tool shows that if caching is enabled for the label’s background, this widget requires 23699 bytes of heap to store the widget. Note that the label is twice the size of the text it contains, so one way of reducing the cost of the widget is to make it smaller, thereby reducing the number of background pixels that must be stored. If the label is resized, the heap allocation is reduced to 11688 bytes, which is a drop of appoximately 50%. Finally, if the background is changed from “Cache” to “Fill” the widget only needs 188 bytes.

The lesson learned is to use Cache as a background only for widgets where it is absolutely necessary and to make the “cached” widgets as small as possible.

Global Palette

Provides information on the Global Palette features.

Description

The Global Palette window is launched from the Graphics Composer’s Asset pull-down menu.

Using a Global Palette enables frame buffer compression for the LCC graphics controller. It creates a 256 color look up table (LUT) and then changes the entire user interface design to adhere to that LUT. Frame buffers are stored as 8 bits/pixel (bpp) indices rather than 16-32 bpp colors. The display driver performs a LUT operation to change each LUT index into a color before writing to the display/controller memory. This enables the use of double buffering, without using external memory, on devices that could not support it before. It also supports single buffering on larger displays. Of course, running the LUT requires more processing on the host. Currently only the LCC graphics controller supports this feature. The Aria demonstration Aria Basic Motion is an example of how using a Global Palette greatly improves the efficiency and capabilities of a design.

Enable the Global Palette by clicking on the Enable Global Palette check box in the window or using the *File > Settings menu*. the Global palette can always be disabled. MHGC will then restore the project back to its original configuration.

If the global palette is enabled you will have to change the MHC configuration of the Graphics Controller to match. For the LCC controller, enable

"Palette Mode". For the GLCD controller, change the Driver Settings > Fame Buffer Color Mode to "LUT8".

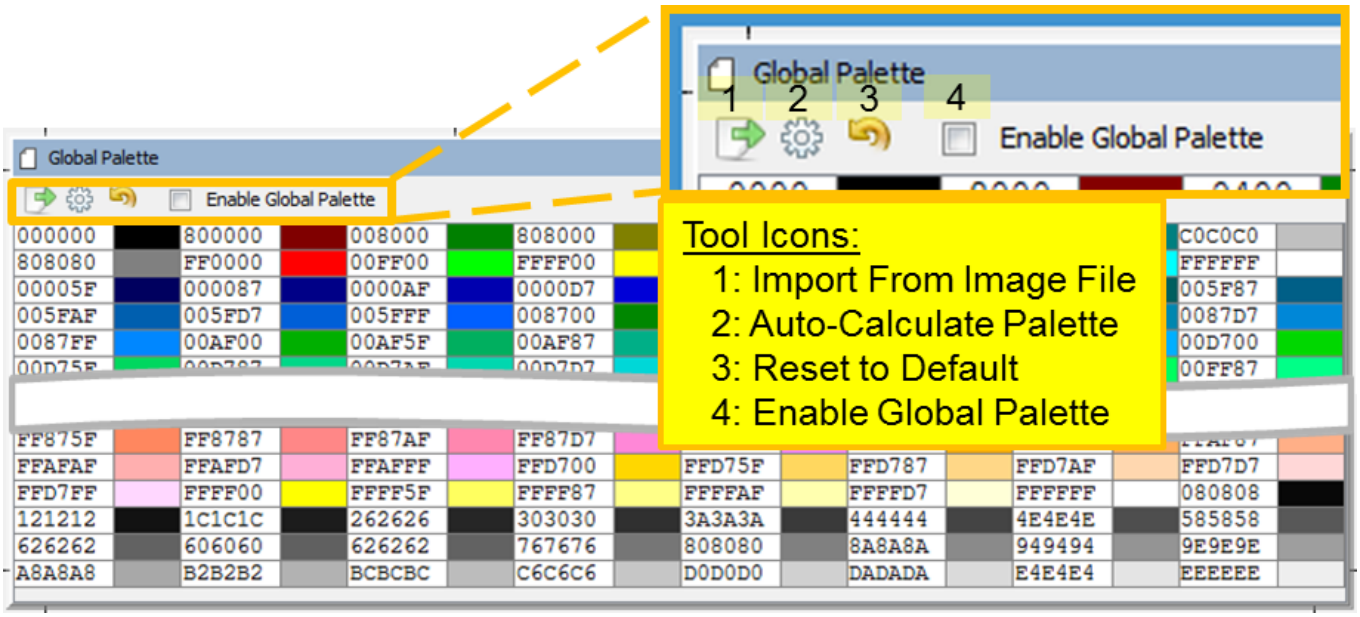
The results of enabling the Global Palette:

- 8bpp frame buffers. In the case of the most common demonstrations this means a 50% reduction in the size of the frame buffer.
- This also opens up the capability to support a single frame buffer for some larger displays.

What is lost by enabling the Global Palette:

- First and foremost - No Dynamic Colors. Dynamic colors are unlikely to match up with an entry in the global palette's look-up table.
- No alpha blending capability. The level of alpha blending can be changed during run-time. (See No Dynamic Colors.)
- No JPEGs or PNGs. Again, no dynamic colors. All images in MGHC will be changed to the color mode of the project, and generated as Raw.
- No font anti-aliasing. Again, no dynamic colors. While the 8-bits/pixel for each glyph is known, the color of the text depends on the color scheme used, and color schemes can change at run time.
- Additional overhead when performing LUT (index->color) operations in the display driver.

The following figure shows the default "Global Palette" when Project Color Mode is set to RGB_888.



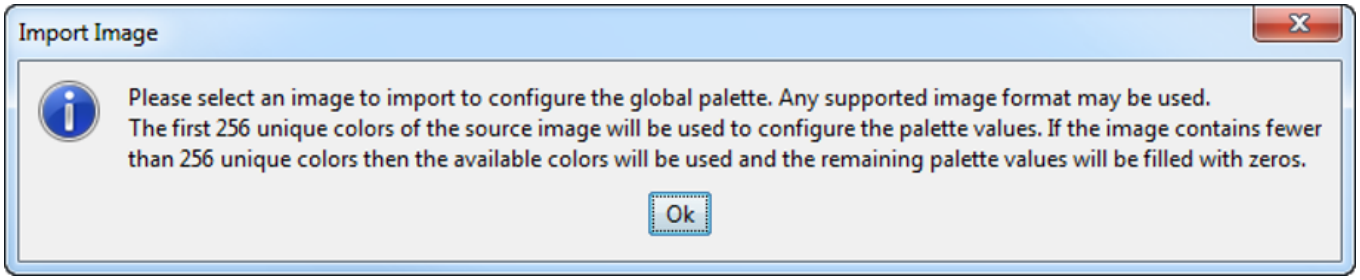
This default palette is good for designs that use a wide array of colors. MHGC also supports developing a custom palette by importing an image defining the palette or by analyzing the pixel colors already in use by the application's images. The palette's color mode is determined by the Project Color Mode, which is determined by the graphics controller.

Clicking on an entry in the palette with bring up the Color Picker dialog window, allowing you to edit the entry's color.

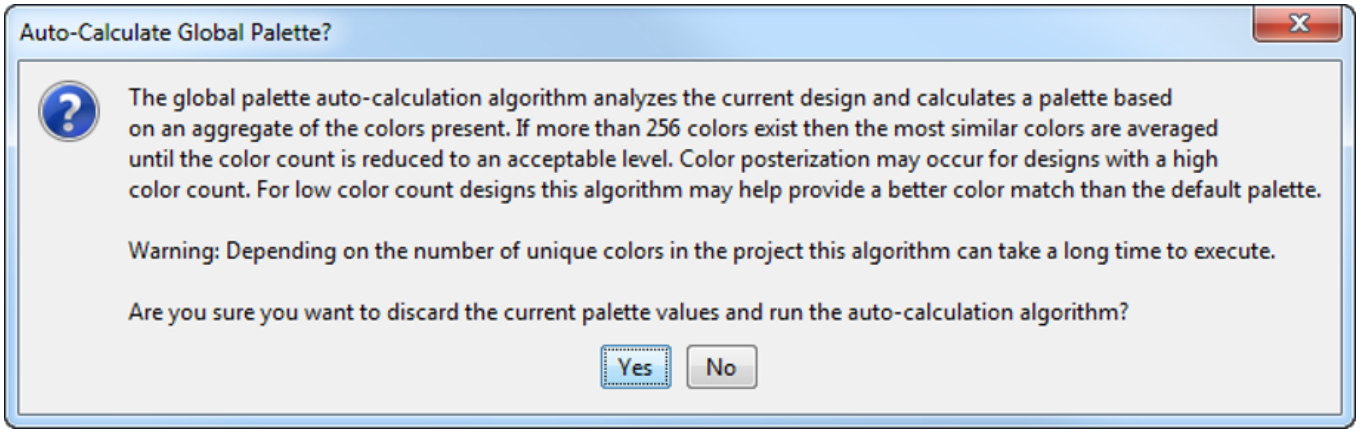
Window Toolbar

There are four icons on the toolbar:

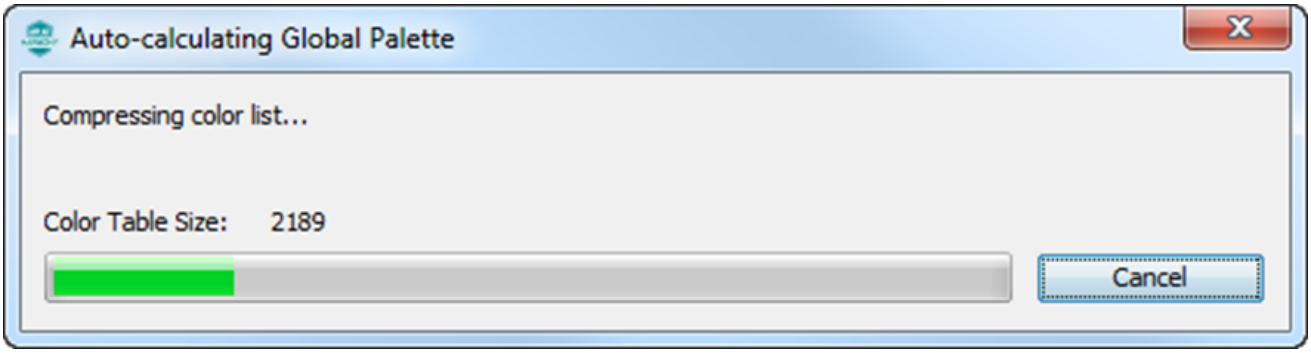
1. **Import From Image File** - Importing a global palette from an image file. Selecting this brings up the following warning. Images can be imported as a BMP,.GIF, JPEG, and PNG (but not TIFF).



2. **Auto-Calculate Palette** – Calculates a new palette using the current design. Selecting this brings up the following warning.

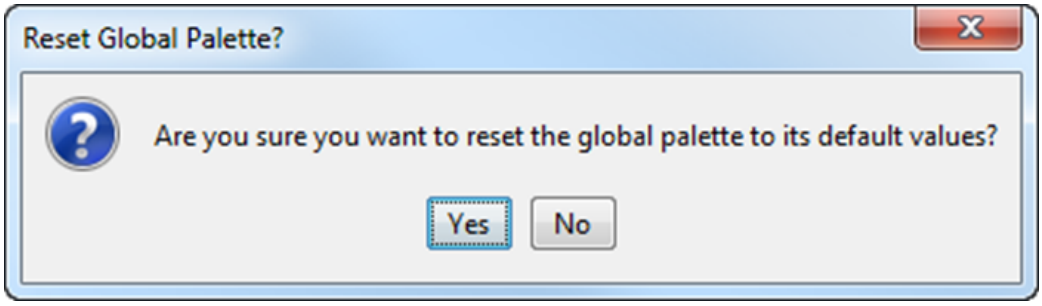


- Selecting **Yes** opens a status window that shows the progress made in selecting a palette of 256 colors

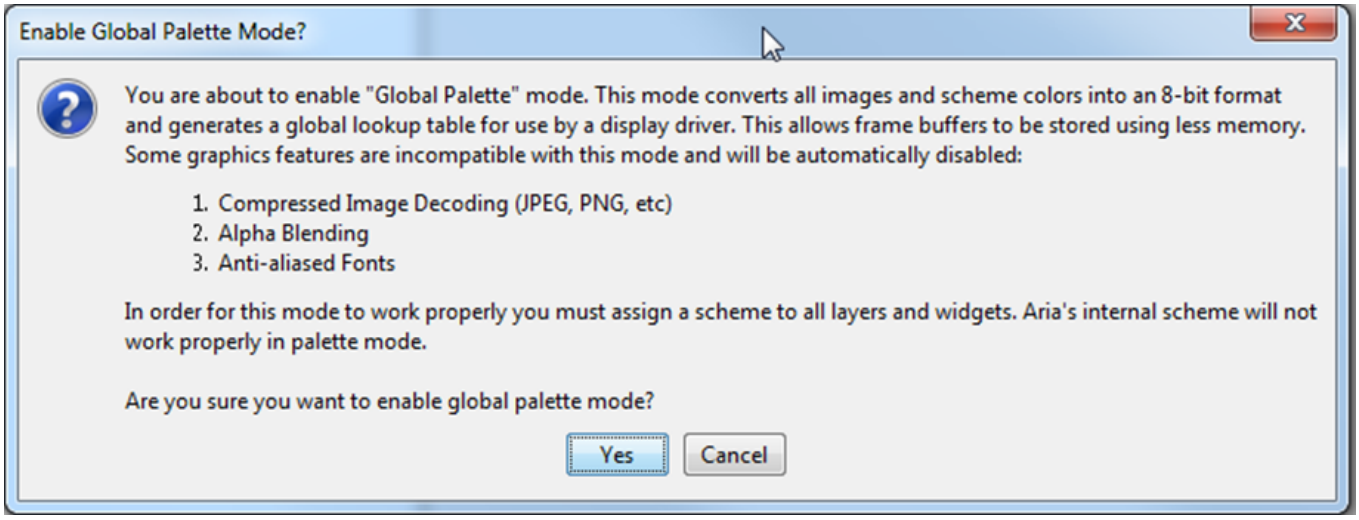


- This can be lengthy operation, but it will effectively generate a palette better tailored to the design. However, extreme (or rare) colors will be changed to nearby, more-plentiful colors, thereby eliminating some of the contrast in images. Whites will tend to darken and blacks lighten. This can be remedied by editing the calculated palette to whiten the whites, darken the blacks, and make other colors closer to the original. This of course may increase the posterization of the image, but that is a natural trade-off in using only 256 colors.

3. **Reset to Default** – This returns the Global Palette to its default values, which opens the Reset Global Palette dialog.



4. **Enable Global Palette** – This performs the same function as *File* > *Settings: Using a Global Palette*. Selecting this opens the Enable Global Palette Mode warning.



Widget Colors

Provides information on widget coloring.

Description

Widget Colors

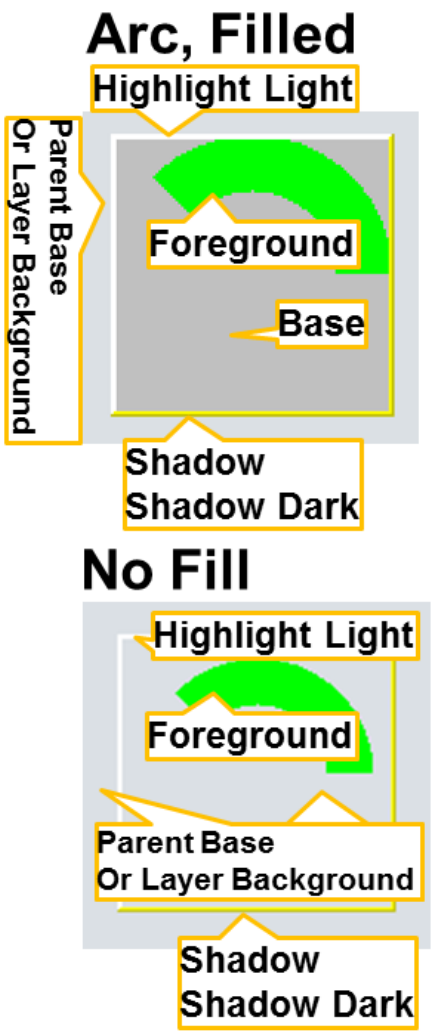
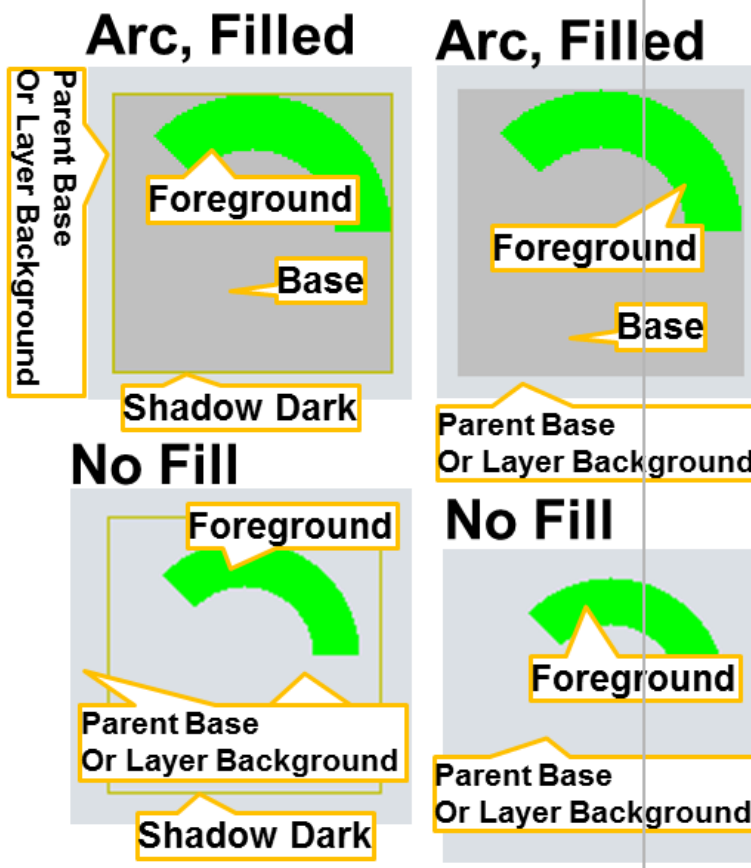
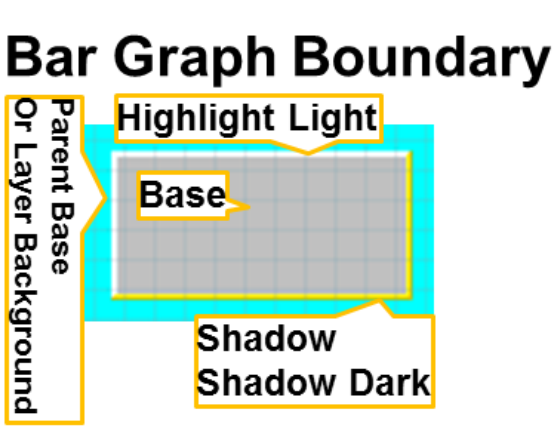
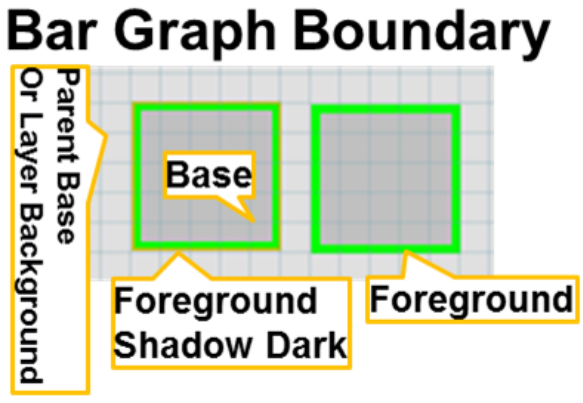
Widget coloring can be customized by creating additional color schemes and assigning these customized schemes to a subset of the widgets uses. For example, a ButtonColorScheme could be customized and used only for Button Widgets.

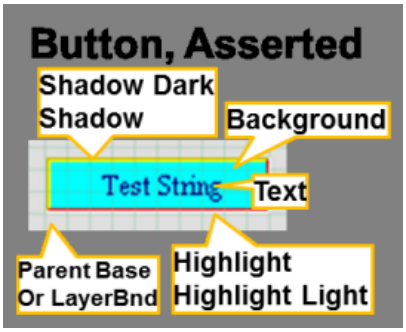
To help highlight the different colors available for each widget, a “CrazyScheme”, with extreme contrast among the 16 available colors, was used as the color scheme for each widget:

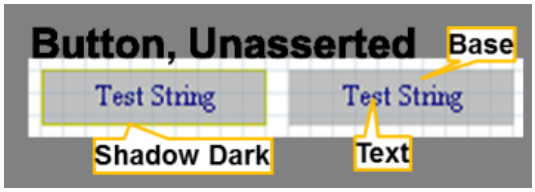
Scheme	
Name	CrazyScheme
Colors	
Base	[192,192,192]
Highlight	[255,0,0]
Highlight Light	[255,255,255]
Shadow	[255,255,0]
Shadow Dark	[192,192,0]
Foreground	[0,255,0]
Foreground Inactive	[0,128,0]
Foreground Disabled	[0,82,0]
Background	[0,255,255]
Background Inactive	[0,192,192]
Background Disabled	[0,128,128]
Text	[0,0,128]
Text Highlight	[0,0,255]
Text Highlight Text	[224,224,224]
Text Inactive	[178,188,191]
Text Disabled	[96,101,102]

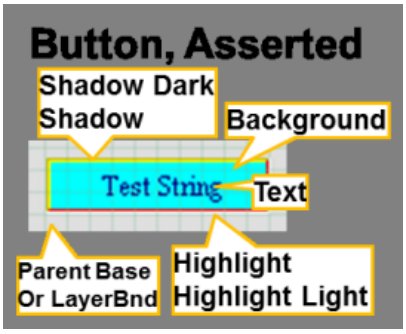
Use this color scheme to help identify the relevant colors for the widgets listed below.

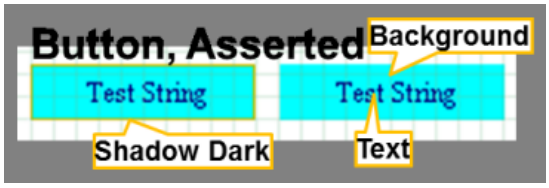
The left column shows the coloring assignments for a Bezel boarder. The right side shows Line/No Border color assignments.

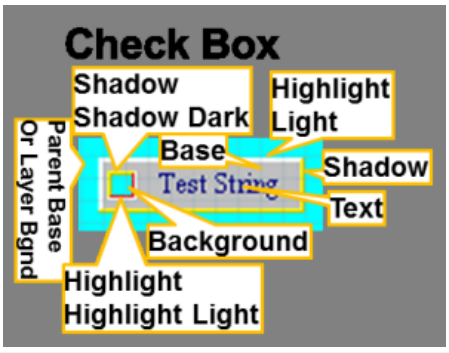
Widget With Bezel Border	Widget With Line or No Borders
Arc Widget: 	
Bar Graph: 	

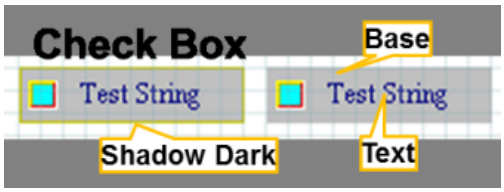

Button, Asserted
Shadow Dark
Shadow
Background
Test String
Text
Parent Base
Or LayerBnd
Highlight
Highlight Light

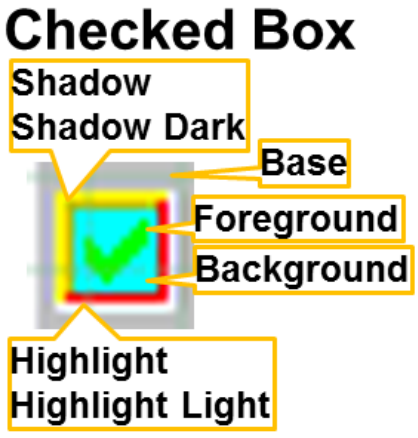

Button, Unasserted
Base
Test String
Test String
Shadow Dark
Text

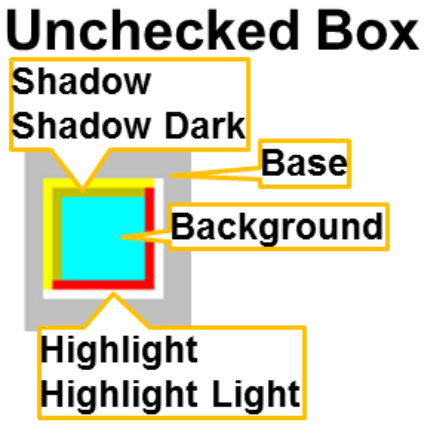

Button, Asserted
Shadow Dark
Shadow
Background
Test String
Text
Parent Base
Or LayerBnd
Highlight
Highlight Light

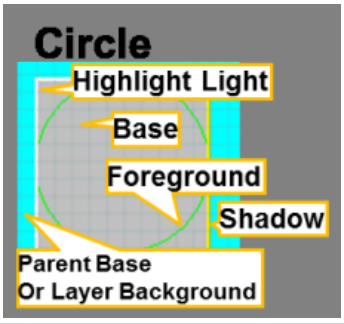

Button, Asserted
Background
Test String
Test String
Shadow Dark
Text

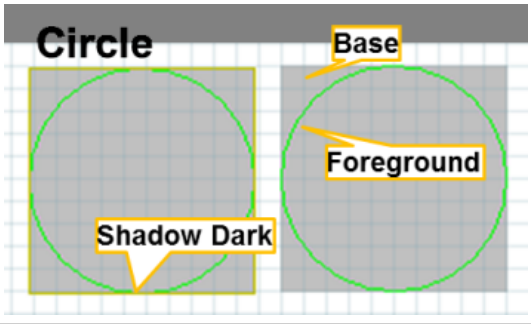

Check Box
Shadow
Shadow Dark
Highlight
Light
Base
Shadow
Test String
Text
Background
Parent Base
Or Layer Bgnd
Highlight
Highlight Light

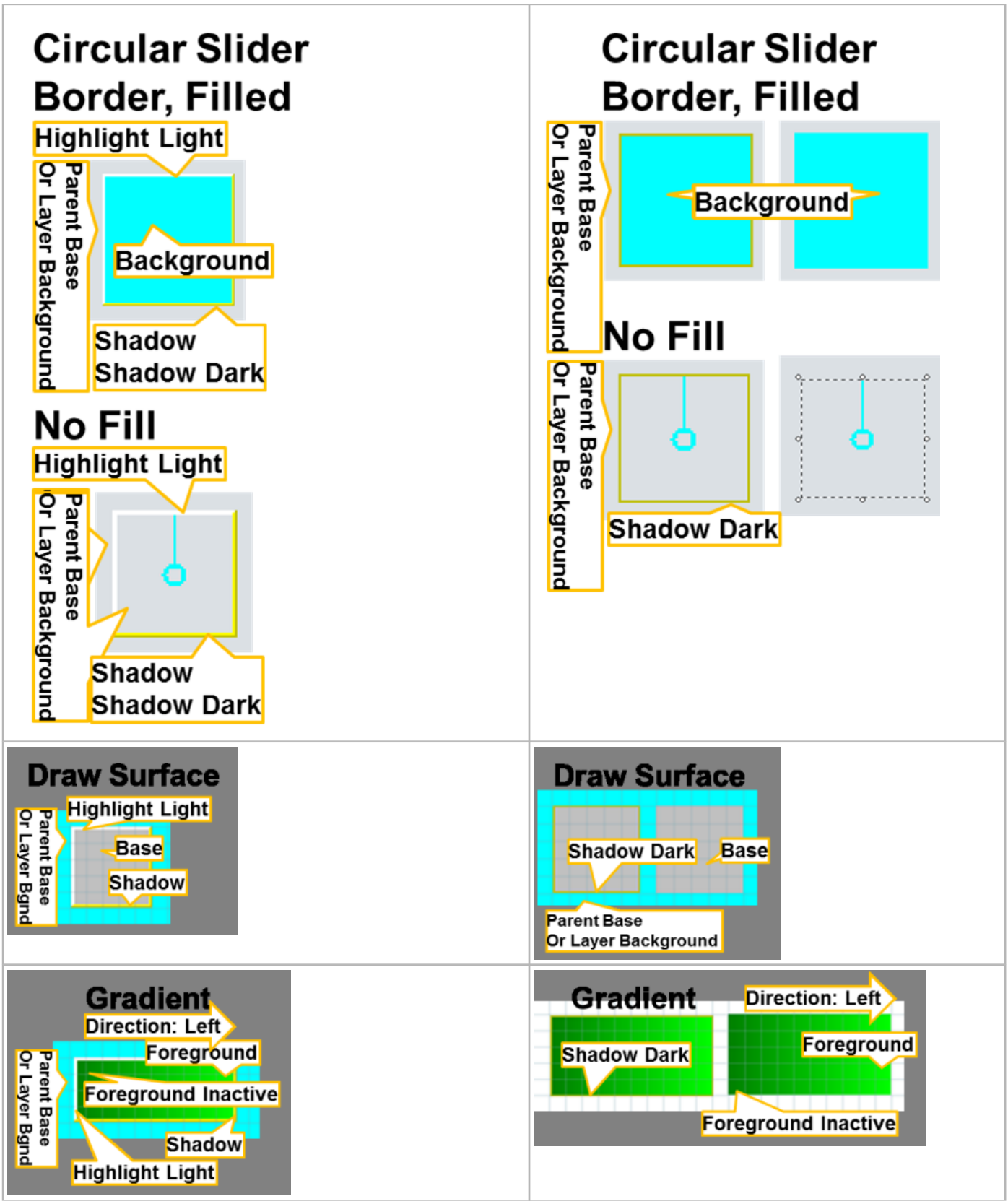

Check Box
Base
Test String
Test String
Shadow Dark
Text

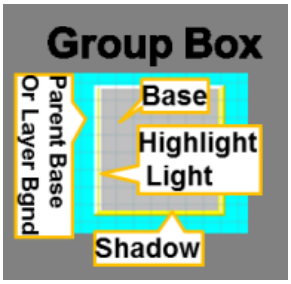

Checked Box
Shadow
Shadow Dark
Base
Foreground
Background
Highlight
Highlight Light

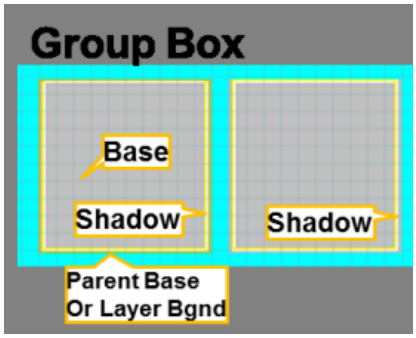

Unchecked Box
Shadow
Shadow Dark
Base
Background
Highlight
Highlight Light


Circle
Highlight Light
Base
Foreground
Shadow
Parent Base
Or Layer Background


Circle
Base
Foreground
Shadow Dark


Circular Slider
Border, Filled
Highlight Light
Parent Base
Or Layer Background
Background
Shadow
Shadow Dark
No Fill
Highlight Light
Parent Base
Or Layer Background
Shadow
Shadow Dark
Circular Slider
Border, Filled
Parent Base
Or Layer Background
Background
No Fill
Parent Base
Or Layer Background
Shadow Dark
Draw Surface
Highlight Light
Parent Base
Or Layer Bgnd
Base
Shadow
Draw Surface
Shadow Dark
Base
Parent Base
Or Layer Background
Gradient
Direction: Left
Foreground
Foreground Inactive
Parent Base
Or Layer Bgnd
Shadow
Highlight Light
Gradient
Direction: Left
Shadow Dark
Foreground
Foreground Inactive


Group Box
Parent Base
Or Layer Bgnd
Base
Highlight
Light
Shadow


Group Box
Base
Shadow
Shadow
Parent Base
Or Layer Bgnd

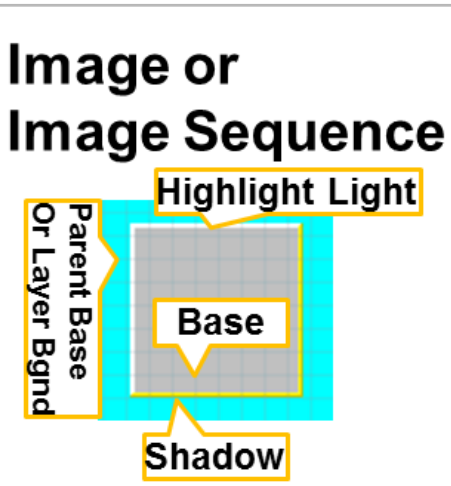
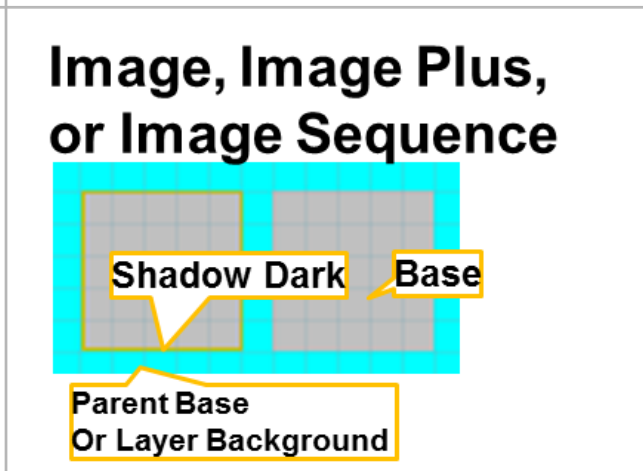
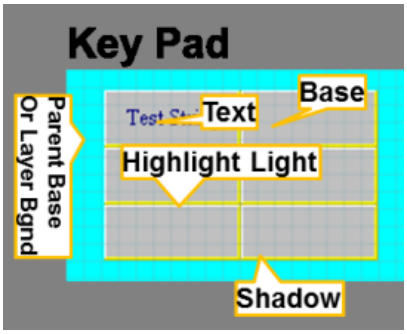
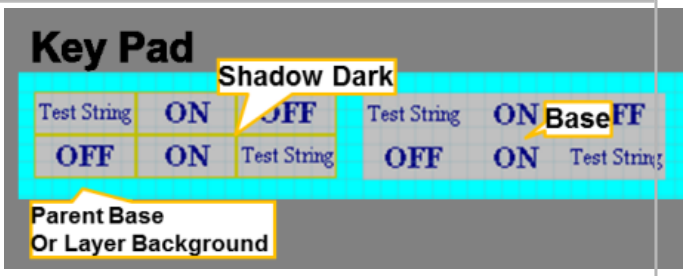
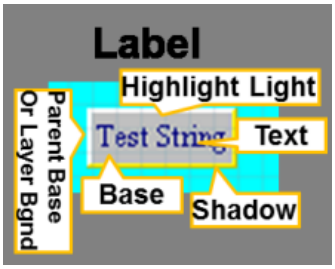


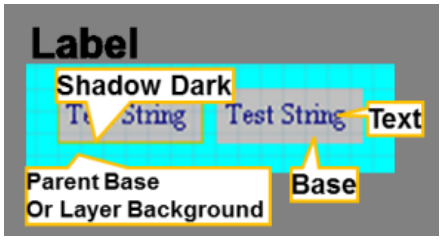
Image or
Image Sequence
Highlight Light
Parent Base
Or Layer Bgnd
Base
Shadow

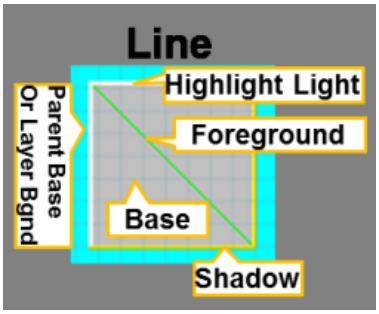

Image, Image Plus,
or Image Sequence
Shadow Dark
Base
Parent Base
Or Layer Background

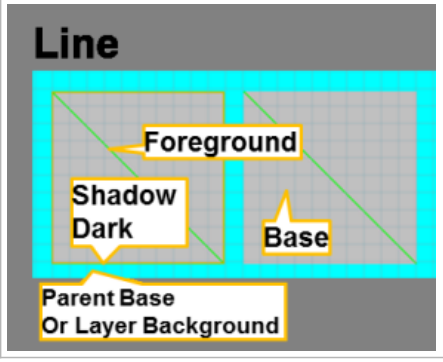

Key Pad
Parent Base
Or Layer Bgnd
Text
Base
Highlight Light
Shadow

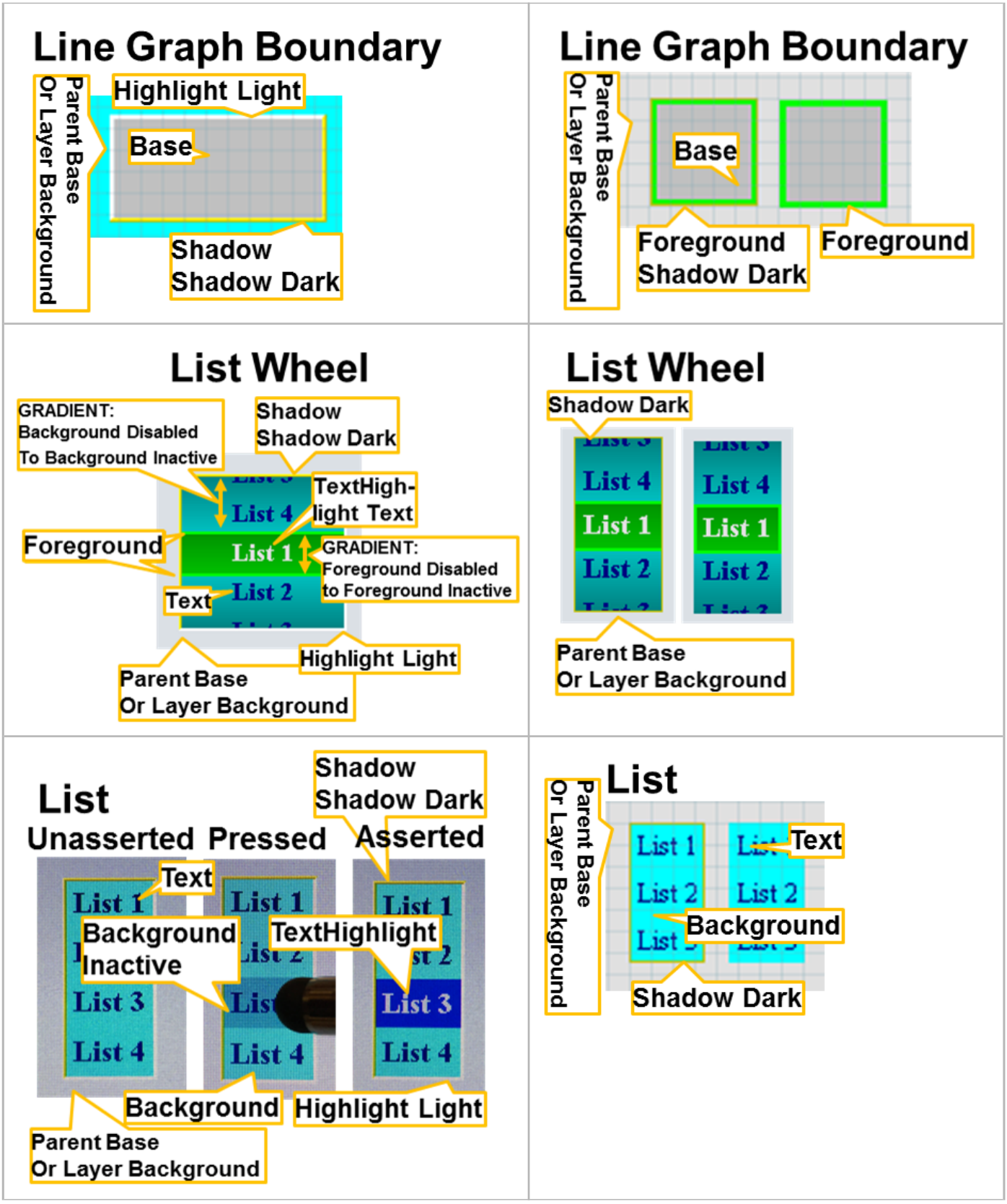

Key Pad
Shadow Dark
Test String
ON
OFF
Test String
ON
Base
FF
OFF
ON
Test String
OFF
ON
Test String
Parent Base
Or Layer Background

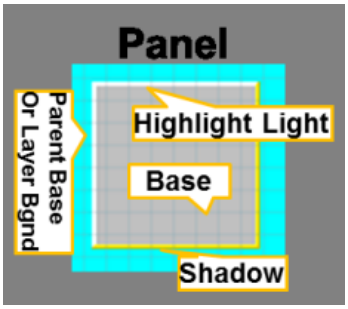

Label
Highlight Light
Parent Base
Or Layer Bgnd
Test String
Text
Base
Shadow

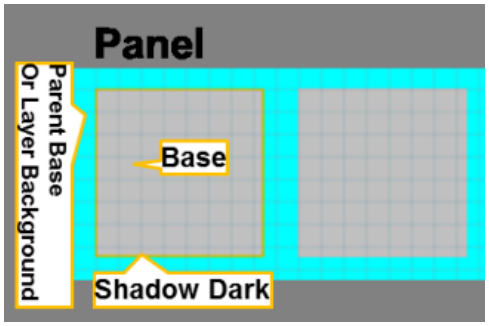

Label
Shadow Dark
Test String
Text
Parent Base
Or Layer Background
Base

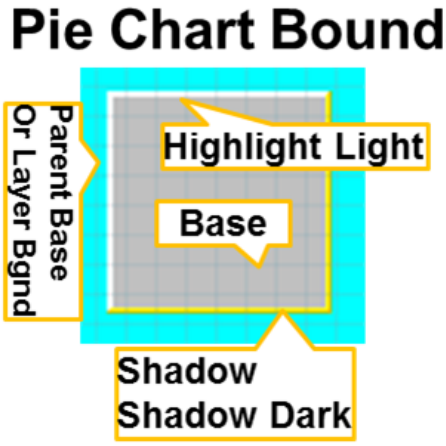

Line
Highlight Light
Parent Base
Or Layer Bgnd
Foreground
Base
Shadow

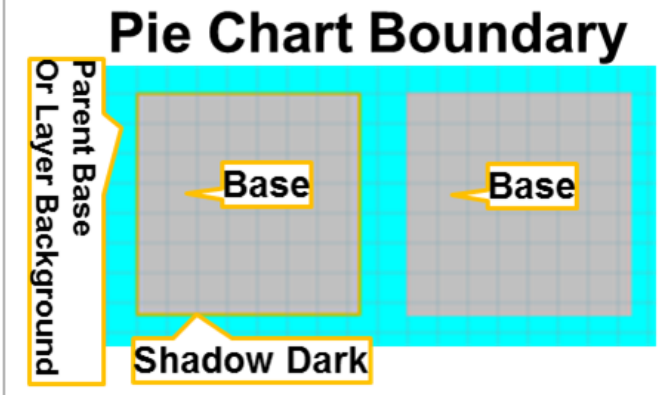

Line
Foreground
Shadow
Dark
Base
Parent Base
Or Layer Background

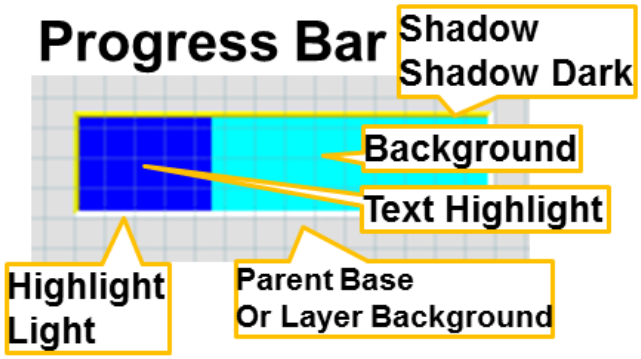

Line Graph Boundary
Parent Base Or Layer Background
Highlight Light
Base
Shadow Shadow Dark
Line Graph Boundary
Parent Base Or Layer Background
Base
Foreground Shadow Dark
Foreground
List Wheel
GRADIENT: Background Disabled To Background Inactive
Shadow Shadow Dark
TextHigh-light Text
List 4
List 1
List 2
Foreground
GRADIENT: Foreground Disabled to Foreground Inactive
Text
Highlight Light
Parent Base Or Layer Background
List Wheel
Shadow Dark
List 4
List 1
List 2
Parent Base Or Layer Background
List
Unasserted Pressed
Shadow Shadow Dark
Asserted
Text
Background Inactive
TextHighlight
List 1
List 2
List 3
List 4
Background
Highlight Light
Parent Base Or Layer Background
List
Parent Base Or Layer Background
List 1
List 2
Text
Background
Shadow Dark

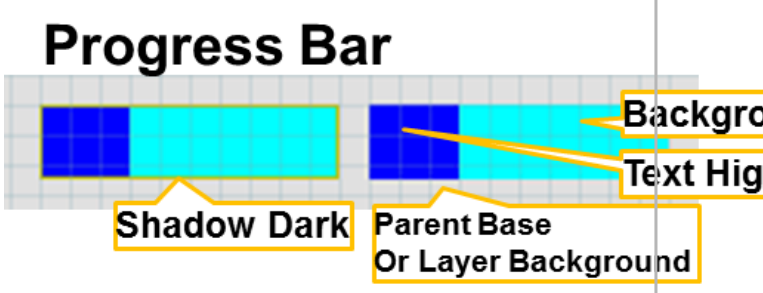

Panel
Parent Base
Or Layer Bgnd
Highlight Light
Base
Shadow

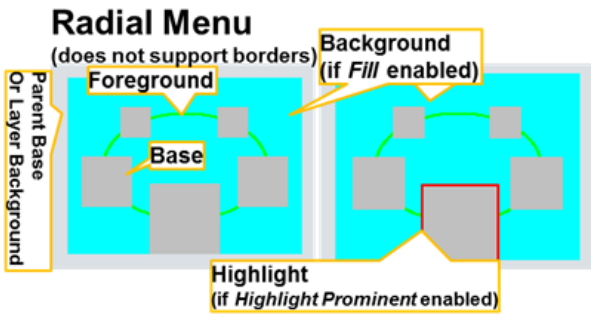

Panel
Parent Base
Or Layer Background
Base
Shadow Dark

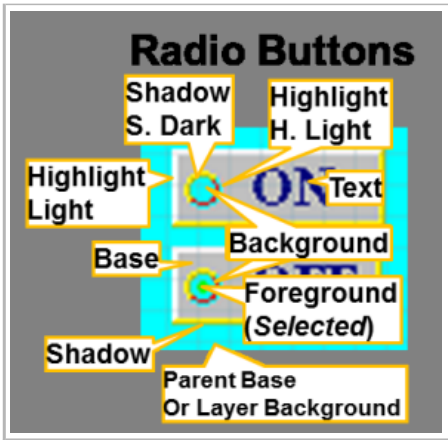

Pie Chart Boundary
Parent Base
Or Layer Bgnd
Highlight Light
Base
Shadow
Shadow Dark

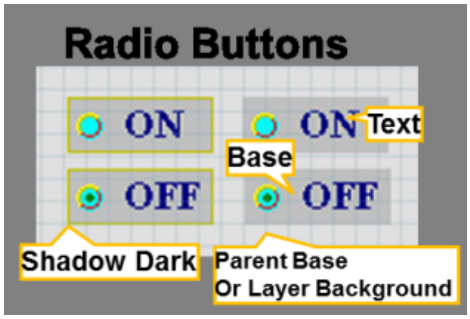

Pie Chart Boundary
Parent Base
Or Layer Background
Base
Base
Shadow Dark

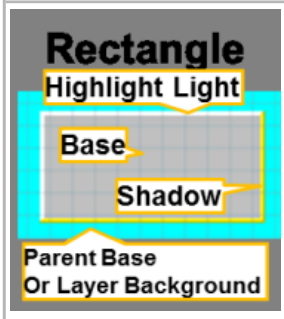

Progress Bar
Shadow
Shadow Dark
Background
Text Highlight
Highlight
Light
Parent Base
Or Layer Background

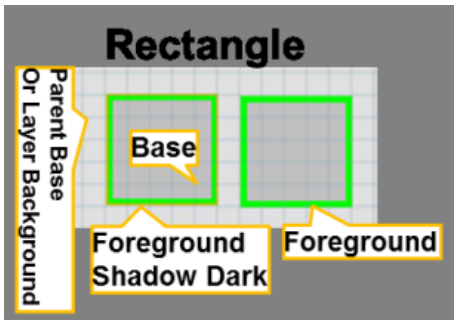

Progress Bar
Backgro
Text Hig
Shadow Dark
Parent Base
Or Layer Background

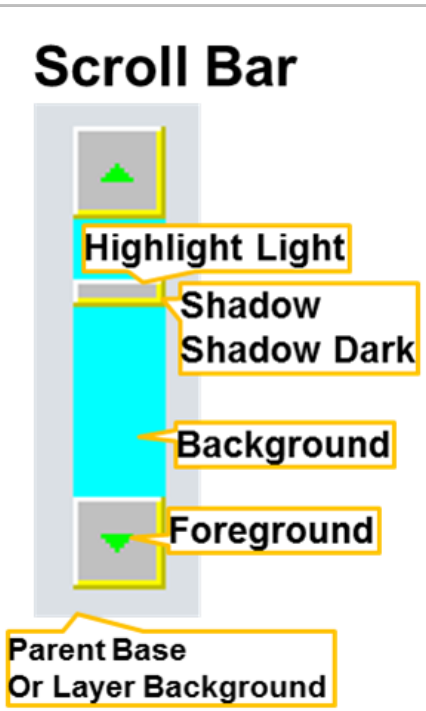

Radial Menu
(does not support borders)
Background
(if Fill enabled)
Foreground
Parent Base
Or Layer Background
Base
Highlight
(if Highlight Prominent enabled)

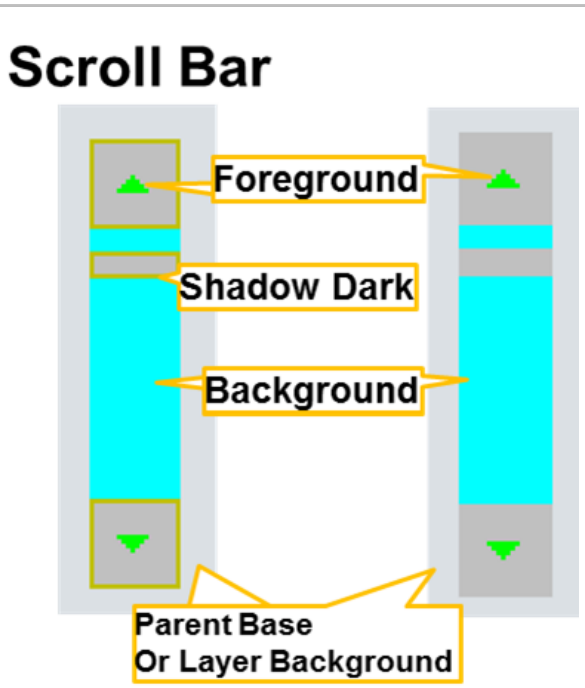

Radio Buttons
Shadow
S. Dark
Highlight
H. Light
Highlight
Light
ON
Text
Background
Base
Foreground
(Selected)
Shadow
Parent Base
Or Layer Background

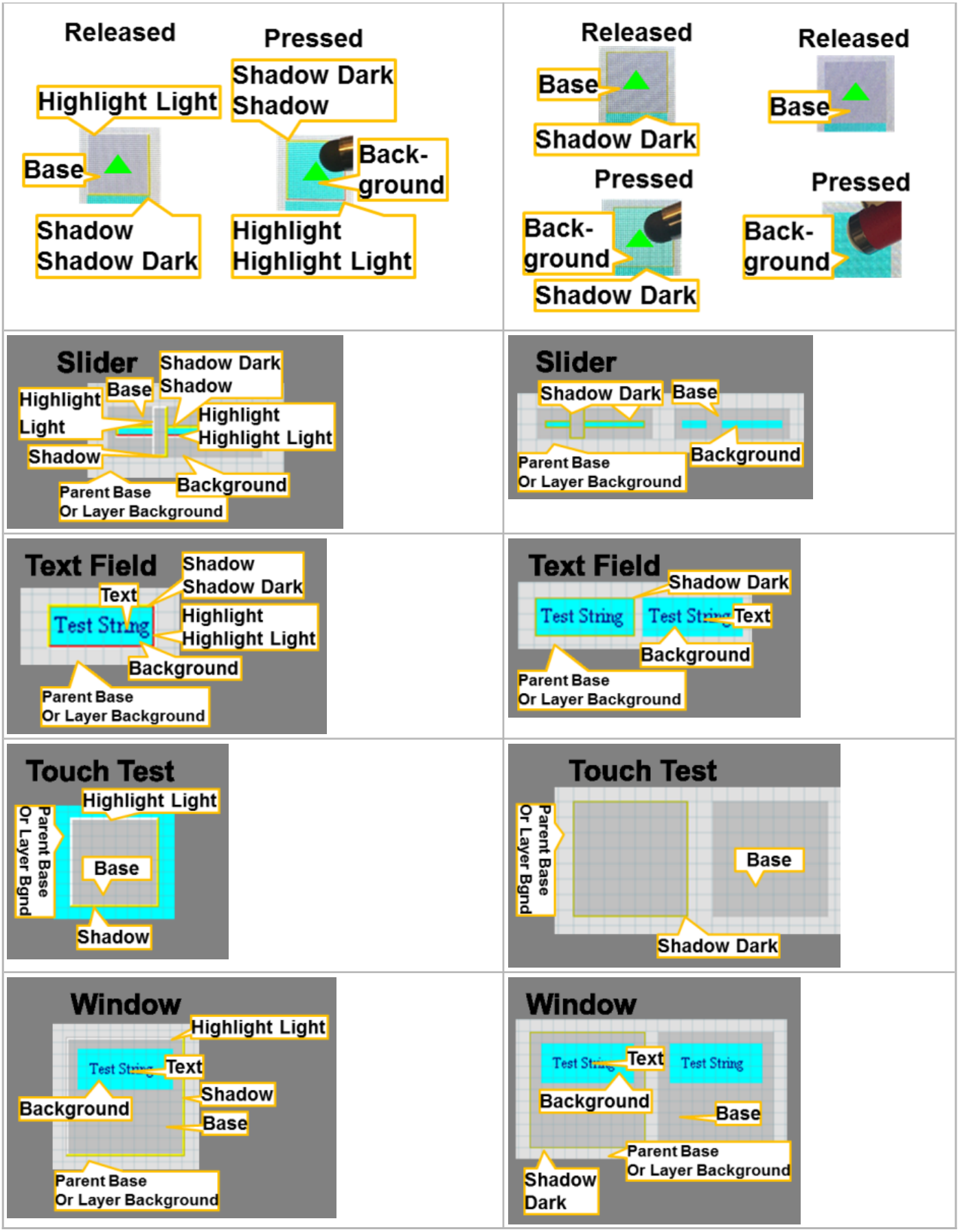

Radio Buttons
ON
ON
Text
Base
OFF
OFF
Shadow Dark
Parent Base
Or Layer Background


Rectangle
Highlight Light
Base
Shadow
Parent Base
Or Layer Background


Rectangle
Parent Base
Or Layer Background
Base
Foreground
Shadow Dark
Foreground


Scroll Bar
Highlight Light
Shadow
Shadow Dark
Background
Foreground
Parent Base
Or Layer Background


Scroll Bar
Foreground
Shadow Dark
Background
Parent Base
Or Layer Background


Released
Pressed
Highlight Light
Shadow Dark
Shadow
Base
Back-
ground
Shadow
Shadow Dark
Highlight
Highlight Light
Released
Base
Shadow Dark
Pressed
Back-
ground
Shadow Dark
Released
Base
Pressed
Back-
ground
Slider
Shadow Dark
Shadow
Base
Highlight
Light
Highlight
Highlight Light
Shadow
Parent Base
Or Layer Background
Background
Slider
Shadow Dark
Base
Parent Base
Or Layer Background
Background
Text Field
Shadow
Shadow Dark
Text
Test String
Highlight
Highlight Light
Background
Parent Base
Or Layer Background
Text Field
Shadow Dark
Test String
Test String
Text
Background
Parent Base
Or Layer Background
Touch Test
Highlight Light
Parent Base
Or Layer Bgnd
Base
Shadow
Touch Test
Parent Base
Or Layer Bgnd
Base
Shadow Dark
Window
Highlight Light
Test String
Text
Background
Shadow
Base
Parent Base
Or Layer Background
Window
Test String
Text
Test String
Background
Base
Parent Base
Or Layer Background
Shadow
Dark

Code Generation

This topic describes using the graphics composer to generate code.

Description

MPLAB Harmony Graphics Composer data is generated the same way as the rest of the project within MHC through the Generate button.

libaria_harmony.h/c – These files provide the interface that binds libaria to the overall MPLAB Harmony framework. They contain the implementations for the standard state management, variable storage, and initialization and tasks functions. If the touch functionality is enabled then the touch bindings are also generated in `libaria_harmony.c`.

libaria_init.h/c - These files contain the main initialization functions for the library state and screens. The header file contains all predefined information for the library state including screen IDs, schemes, and widget pointers. The main initialization function initializes all schemes and screens, creates all screen objects, and sets the initial state of the library context. As each screen must be capable of being created at any time, each screen has a unique create function that can be called at any time by the library. The `libaria_init.c` file contains these create functions.

libaria_events.h/c – The event files contain the definitions and implementations of all enabled MHGC events. Each event implementation will contain all generated actions for that event.

libaria_macros.h/c – The macro files contain the definitions and implementations of all defined MHGC screen macros. A macro is similar to an event in that it can contain actions. However, it is meant to be called from an external source such as the main application.

libaria_config.h – This file contains configuration values for the library. These are controlled through settings defined in the MHC settings tree.

gfx_display_def.c – This file contains generated definitions for enabled graphics displays.

gfx_driver_def.c – This file contains generated definitions for enabled graphics drivers.

gfx_processor_def.c – This file contains generated definitions for enabled graphics processors.

gfx_assets.h/c – These files contain generated asset data.

Advanced Topics

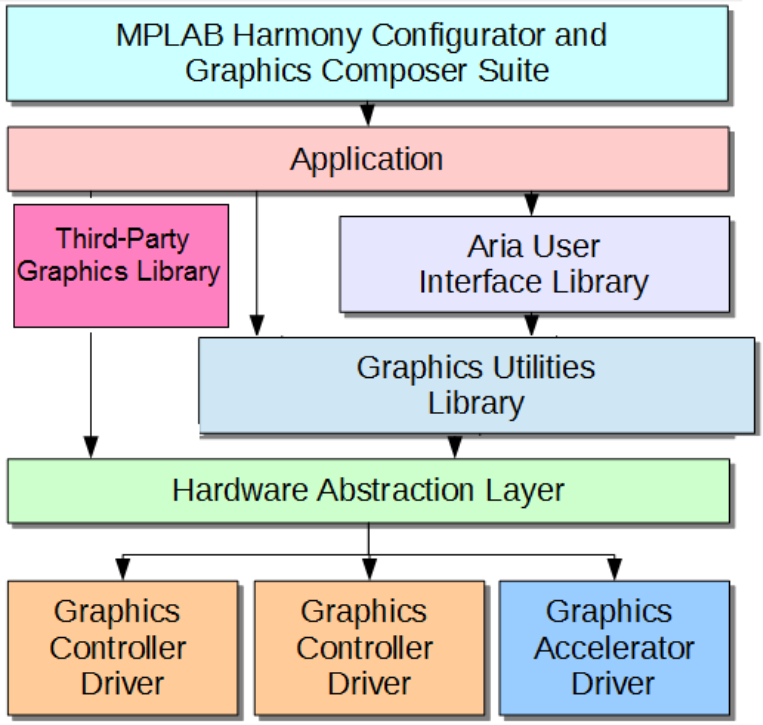
This section provides advanced information topics for MHGC.

Adding Third-Party Graphics Products Using the Hardware Abstsraction Layer (HAL)

This topic provides information on using the Hardware Abstraction Layer (HAL) to add third-party graphics products.

Description

The architecture of the MPLAB Harmony Graphics Stack is shown in the following diagram.



Hardware Abstraction Layer (HAL)

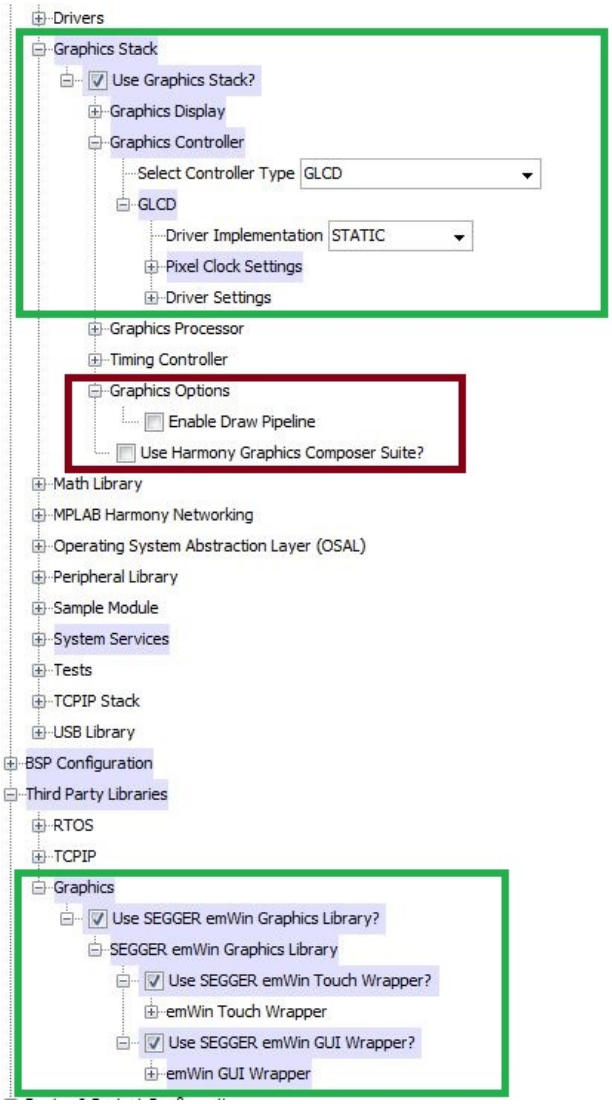
The HAL is a software layer that serves as a gatekeeper for all graphics controller and accelerator drivers. This layer is configured at initialization by the underlying graphics drivers and provides functionality such as buffer management, primitive shape drawing, hardware abstraction, and draw state management. This layer serves as a means of protection for the drivers, frame buffers, and draw state in order to prevent state mismanagement by the application.

Third-Party Graphics Library

The third-party graphics library can be used with the MPLAB Harmony framework to perform the graphics operations desired by the application. The third-party library has access to the HAL, which has been configured to service the frame buffer which is filled by the third-party graphics library.

The third-party graphics library can access the MPLAB Harmony framework drivers such as touch drivers, graphics controller driver, and display driver through the HAL. The draw pipeline and the user interface (UI) design files come from the third-party graphics library. The third-party graphics library needs the frame buffer location to fill the frame buffer with the pixel values. Or, in case of external controllers, it would need a function to access the controller drivers to output pixels on the display. The HAL provides the third-party graphics library with the frame buffer location or the API to communicate the pixel values to the external controllers.

The following figure from the MPLAB Harmony Configurator (MHC), shows the selections made in the Graphics Stack to enable the needed graphics display and controller features. Note that the Draw Pipeline for the MPLAB Harmony Graphics Stack has been disabled to assure that the third-party graphics alone is taking effect. The MPLAB Harmony Graphics Configurator (MHGC) is also not enabled, as the design tools from the third-party graphics library are used to generate the UI graphics. The `LCDConf.c` file has appropriate APIs for the third-party graphics library to communicate through the HAL with the display drivers and the framebuffer.



Example Demonstration Project

The Aria demonstration project, emwin_quickstart, has three configurations. Each configuration has an API named LCD_X_Config, which is

generated with the relevant calls for SEGGER emWin to communicate with the display driver and obtain the frame buffer location pointer to write the pixel data to it. For PIC32MZ DA and PIC32MZ EF configurations, the frame buffer pointer address is provided to SEGGER emWin by the HAL. For the S1D controller on PIC32MX devices (pic32mx_usb_sk2_s1d_pictail_wqvga), The pixel write function pointers are assigned to the appropriate S1D driver APIs, which allow SEGGER emWin to write to the display controller.

Speed and Performance of Different Image Decode Formats in MHGC

Provides information and recommendations for image decode formats.

Description

MHGC supports various image formats and the MHGC Image Assets Manager provides the ability to convert and store a source image into to the following formats

- Bitmap RAW
- Bitmap Raw Run-Length Encoded (RLE)
- JPEG
- PNG
- Predecoded RAW Bitmap in DDR (PIC32MZ DA)

The following table shows the relative rendering time and Flash memory requirements of the different image formats in the MPLAB Harmony Graphics Library. The rendering time includes decoding the image and drawing it to the screen. This information is helpful when optimizing a MPLAB Harmony graphics project for performance and/or Flash memory space. For example, as shown by the red highlighted text in the table, a 40x40 pixel 16-bit RAW image renders 2.38 times faster and uses 2.59 times more Flash space than a JPEG image.

Image Format	Resolution (Pixels)	Size In Flash (Bytes)	Relative Frame Update Rate		Relative Size In Flash	
			Versus RAW (16-bit)	Versus JPEG (24-bit)	Versus RAW (16-bit)	Versus JPEG (24-bit)
Raw 16-bit	***40x40***	***3200***	***1***	***2.38***	***1***	***2.59***
	100x100	20000	1	2.73	1	6.23
	200x200	80000	1	2.67	1	11.23
Raw 16-bit RLE	40x40	1796	0.71	1.68	0.56	1.45
	100x100	9288	0.57	1.55	0.46	2.89
	200x200	29916	0.56	1.5	0.37	4.2
JPG (24-bit)	40x40	1237	0.42	1	0.39	1
	100x100	3212	0.37	1	0.16	1
	200x200	7123	0.38	1	0.09	1
PNG (32-bit)	40x40	1999	0.34	0.8	0.62	1.62
	100x100	6782	0.25	0.68	0.34	2.11
Predecoded RAW in DDR (from JPG)	40x40	1237	0.81	1.93	0.39	1
	100x100	3212	2.68	7.32	0.16	1
	200x200	7123	10.06	26.83	0.09	1

Predecoded Images in DDR (RAW)

For PIC32MZ DA devices with DDR, the MHGC Image Asset Manager provides an option to predecode images from Flash and store them into DDR as RAW images. The GPU is used to render the decoded image from DDR to the frame buffer. This provides a faster render time than an equivalent RAW image in Flash memory, specifically for large images (up to 10 times faster for a 200x200 image). Conversely, predecoding small images 40x40 pixels or smaller in DDR may not render faster due to the additional overhead of setting up the GPU.

Recommendations:

- If there is adequate DDR memory available, consider predecoding images to DDR for best performance
- Using JPEG images and predecoding them into DDR can provide the best rendering performance and most Flash memory savings.

Note: The images are decoded from Flash to DDR memory by the Graphics Library during initialization and may introduce delay at boot-up, depending on the number and size of the images.

RAW Images

RAW images provide fast rendering time, as there is no decoding needed. However, depending on image content, it can be two times larger than a Run-Length Encoded (RLE) image and about 3 to 10 times larger than a JPEG.

Recommendation:

For small images that are to be rendered frequently, consider using a RAW image for better performance

JPEG Images

JPEG images provide the most Flash space savings, but are slower to render compared to RAW and RAW RLE.

Recommendations:

- If images are large and not used frequently, consider using the JPEG image format to save flash memory space
- If DDR memory is available, consider predecoding JPEG images in DDR for better rendering performance

Run-Length Encoded RAW Images

In terms of rendering speed and size, RAW RLE images are in between RAW and other compressed formats like JPEG or PNG. Depending on the image contents, RAW RLE can be approximately 1.5 times faster than JPEG, but could be significantly larger in size for large images. Again, depending on the image content, RAW RLE can be about half the size and performance of a RAW image.

Recommendation:

If optimizing your application for both speed and flash size consider using RAW RLE images

PNG Images

Among the image formats, PNG is slowest to render and requires more memory to decode.

Recommendations:

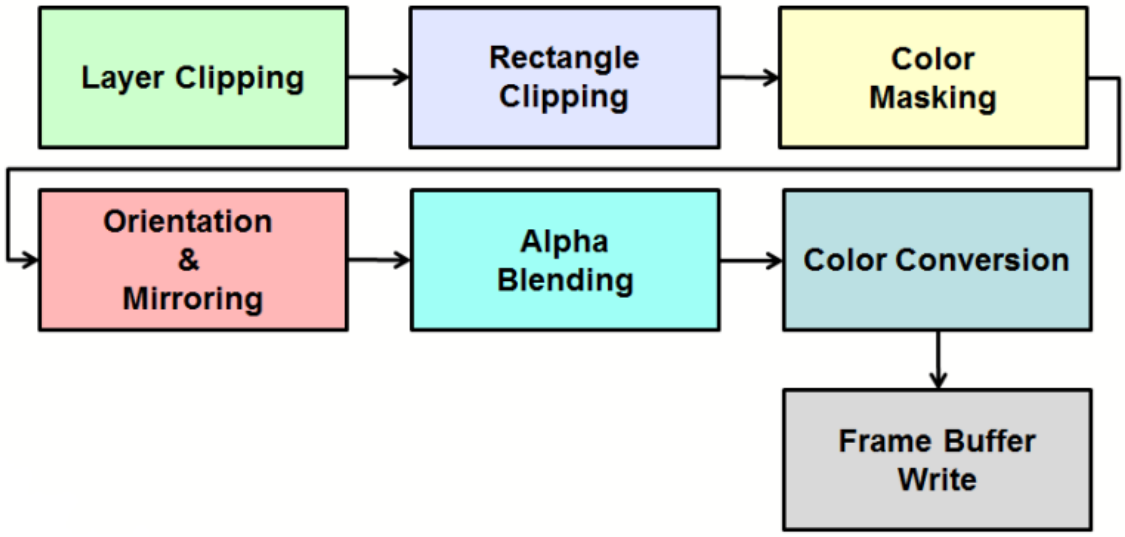
- Unless fine levels of alpha-blending are needed, it is better to use other image formats to achieve the best performance. Use the MHGC Asset Manager to convert the source PNG image and store it in a different image format.
- If you would like to use an image with a transparent background, it may be better to use a RAW RLE image with background color masking to achieve the same effect with better performance than a PNG. Color masking is supported in the MHGC Image Asset Manager.

Draw Pipeline Options

This section details how to use the Graphics Pipeline.

Description

The nominal rendering pipeline for an image is shown in the following figure.



The order of rendering for other widgets may differ. For example, for a colored rectangle the color mask is first checked. If the rectangle's fill matches the mask color defined then there is nothing to draw.

Graphics Pipeline

Provides information on the graphics pipeline.

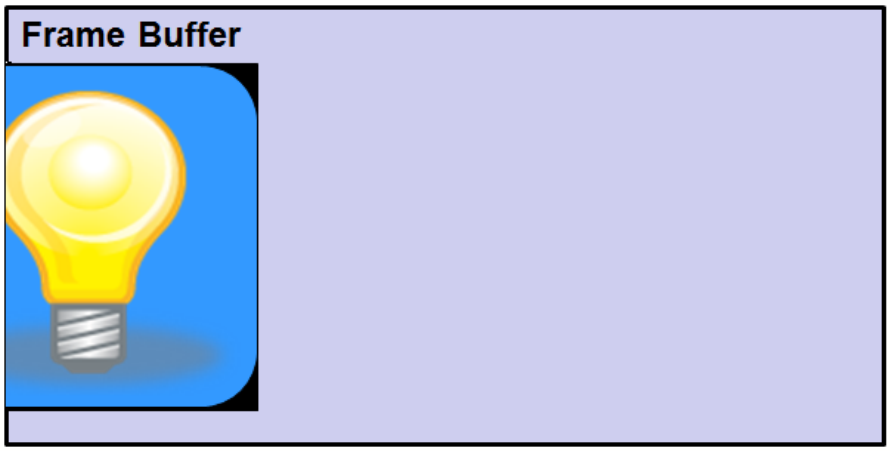
Description

Layer Clipping

In order of the processing, Layer Clipping is first applied to the image. If the image extends beyond the edges of the layer that contains it then those pixels are not drawn. Failure to clip out-of-bound pixels can cause the application to crash. The following figures shows an example of layer clipping:

Before applying layer boundaries:

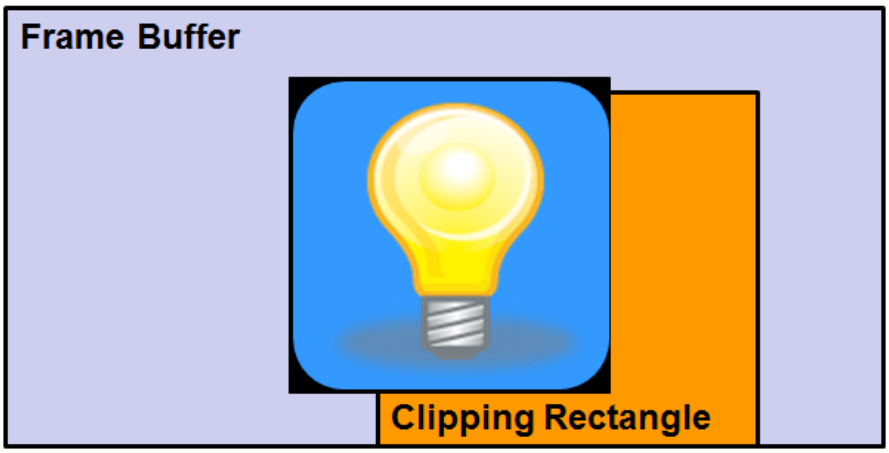
After applying layer boundaries:



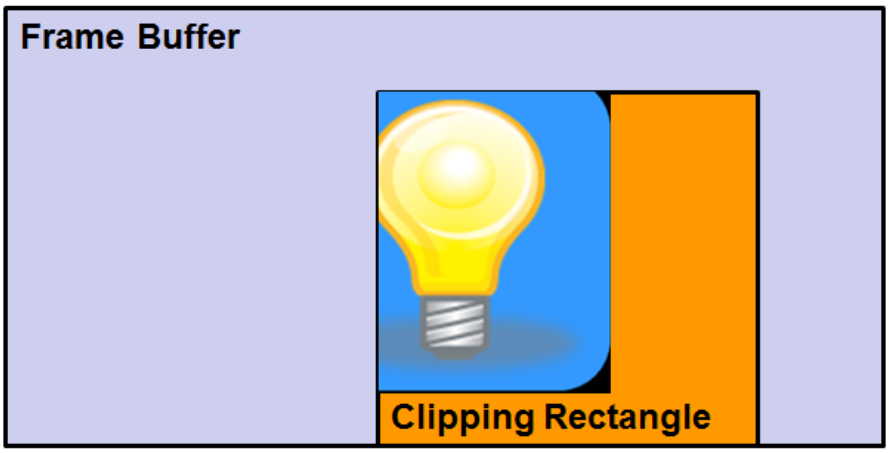
Rectangle Clipping

Next, the image is clipped to the boundaries of any widgets that contain it as a parent, such as a rectangle.

Before applying the clipping rectangle.:



After applying the clipping rectangle:



Color Masking of Pixels

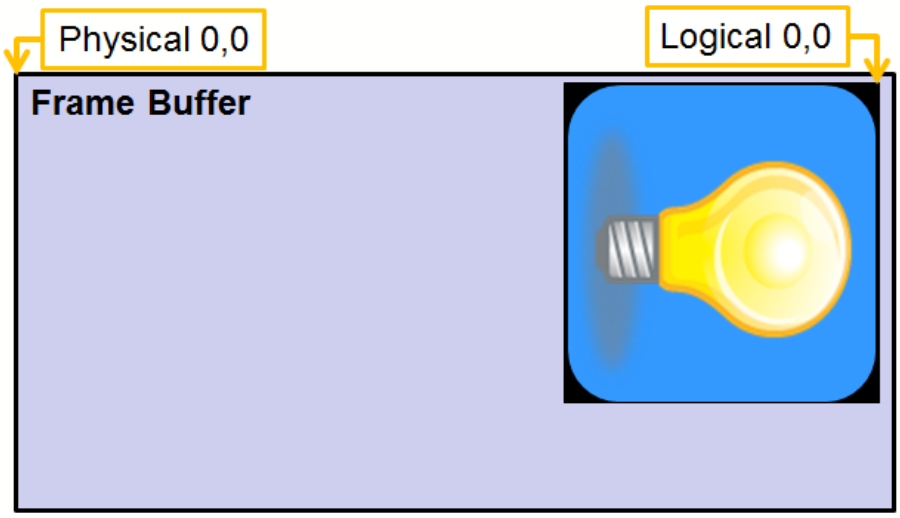
Pixels in the image are matched to a mask color. If the colors match the pixel is discarded (not drawn). In the following example, the black border of the image is removed by defining the mask color to be black.

Before applying color mask:

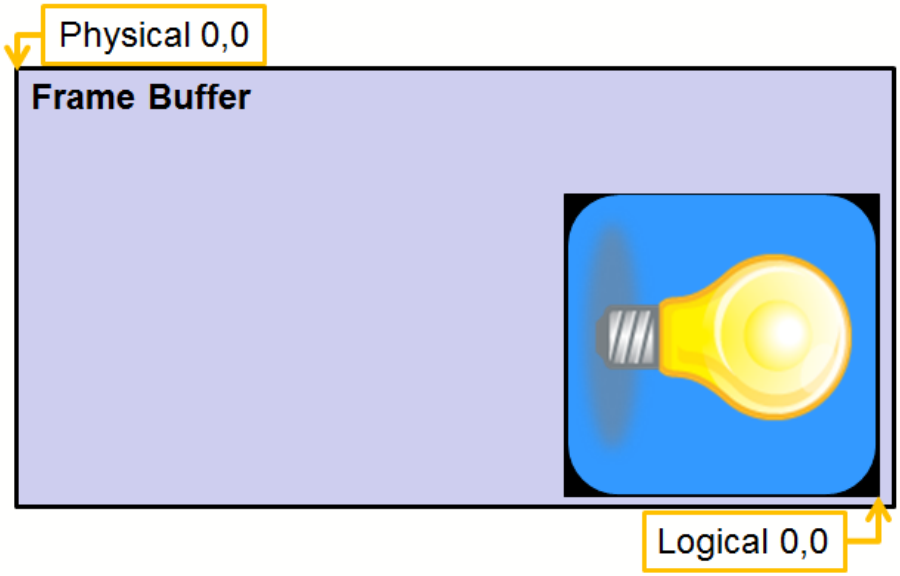
After applying color mask:

Orientation and Mirroring

The logical orientation of the graphics design may not match the physical layout of the display. Pixels may need to be reoriented from logical to physical space before being rendered.



Pixels may also need to be flipped (mirrored) before being rendered.

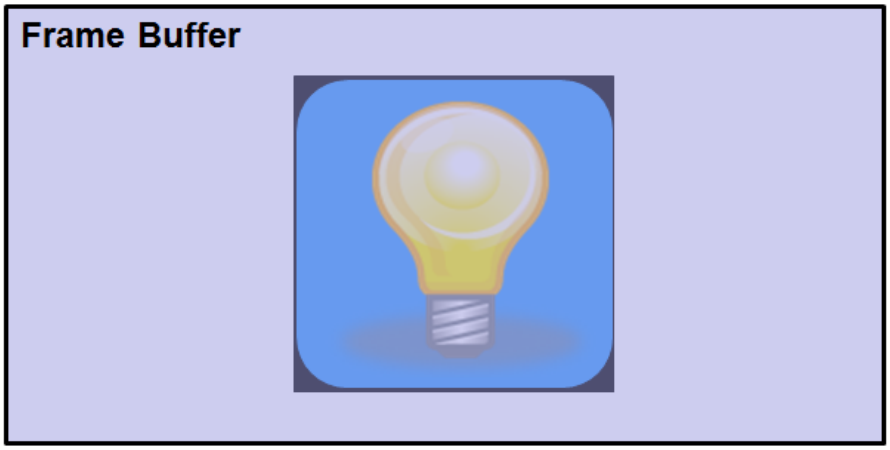


Alpha Blending

Each pixel drawn is a composite of the image color and the background color based on the alpha blend value defined by a global alpha value, the pixels alpha value, or both.

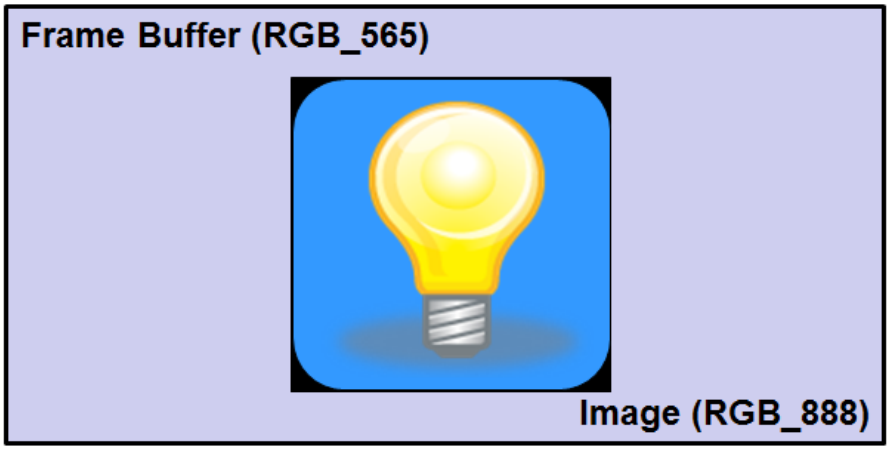
Before alpha blending:

After alpha blending:



Color Conversion

The image color format may not be the same as the destination frame buffer. Each pixel must be converted before it is written. In the following example, the image is stored using 24 bits per pixel; however, the frame buffer uses 16 bits per pixel.



Frame Buffer Write

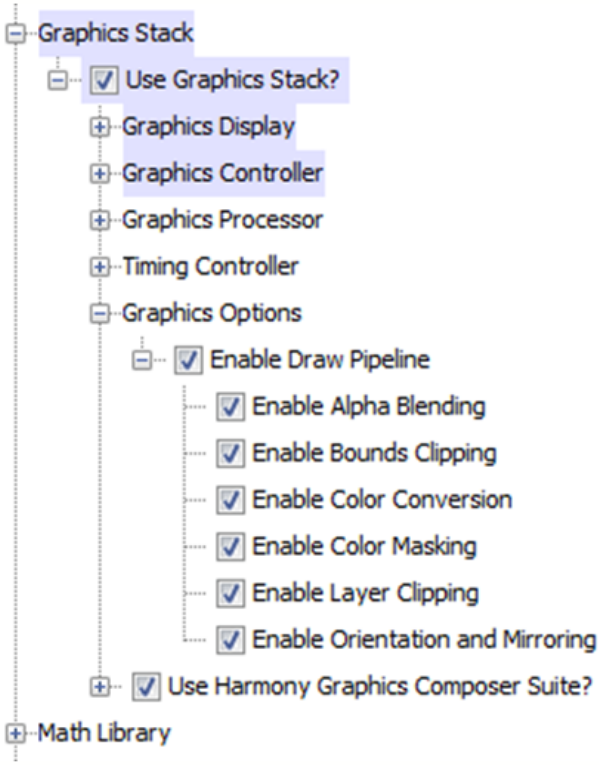
The final stage in rendering an image is to write each-color converted pixel to the frame buffer.

Graphics Pipeline Options

Provides information graphics pipeline options.

Description

Each stage in the graphics pipeline adds overhead to the rendering. Stages can be removed from processing using MPLAB Harmony Configurator (MHC) options for the Draw Pipeline, found by selecting *MPLAB Harmony Framework Configuration* > *Graphics Stack*.



For example, the Alpha Blending stage can be disabled if your graphics application does not use alpha blending. If the color mode of the display matches the color mode of all images you can disable Color Conversion. Disabling unneeded stages can improve performance and reduce code size.

Also, a graphics controller driver may add additional stages, or opt to bypass stages completely depending on the capabilities of the graphics hardware supported by the driver.

Improved Touch Performance with Phantom Buttons

This topic provides information on the use of phantom buttons to improve touch performance.

aria_coffeemaker Demonstration Example

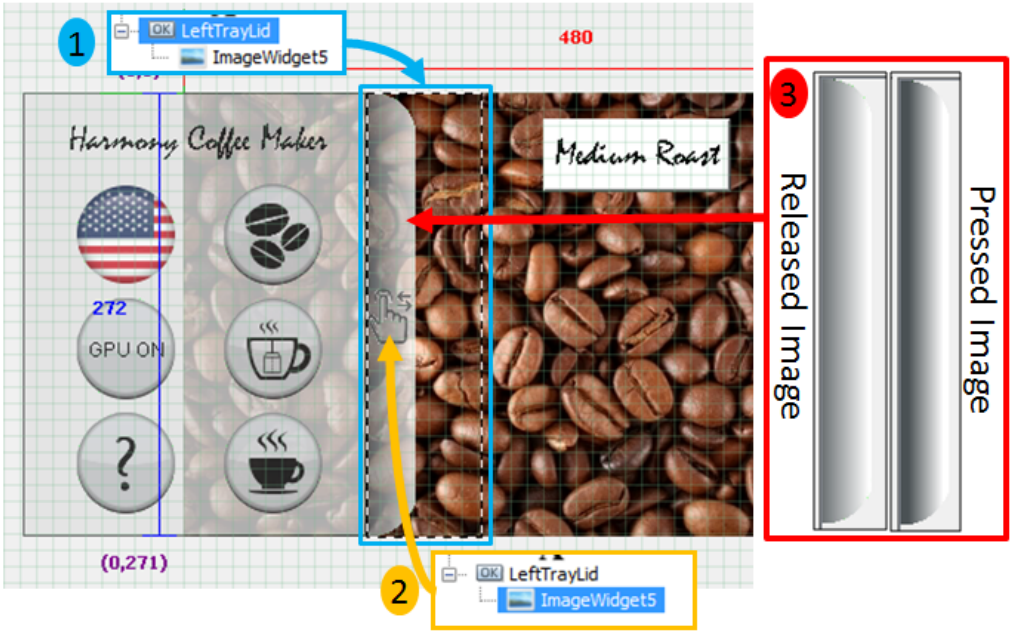
Provides image examples with buttons in the aria_coffeemaker demonstration.

Description

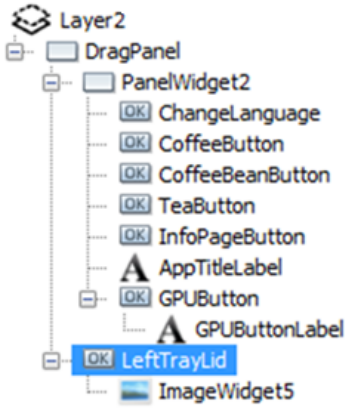
Small buttons are hard to activate on the screen. The use of phantom (invisible) buttons can improve touch performance without increasing the size of the visible footprint of the button on the display.

The aria_coffee_maker has a sliding tray on each side of the display. Sliding a tray in, or out, is accomplished by a phantom (invisible) button. Looking at the left tray, we see the three parts of this phantom button.

1. *LeftTrayLid*: An invisible button widget, whose outline is shown in blue. This area is the touch field.
2. *ImageWidget5*: An image widget containing a hand icon, providing a visual clue as to how to manipulate the tray.
3. *The Release Image and Pressed Image*: These are defined as part of the button widget properties. The Pressed Image has a darker coloring than the Released Image. This difference is what shows the user that the button has been pressed.

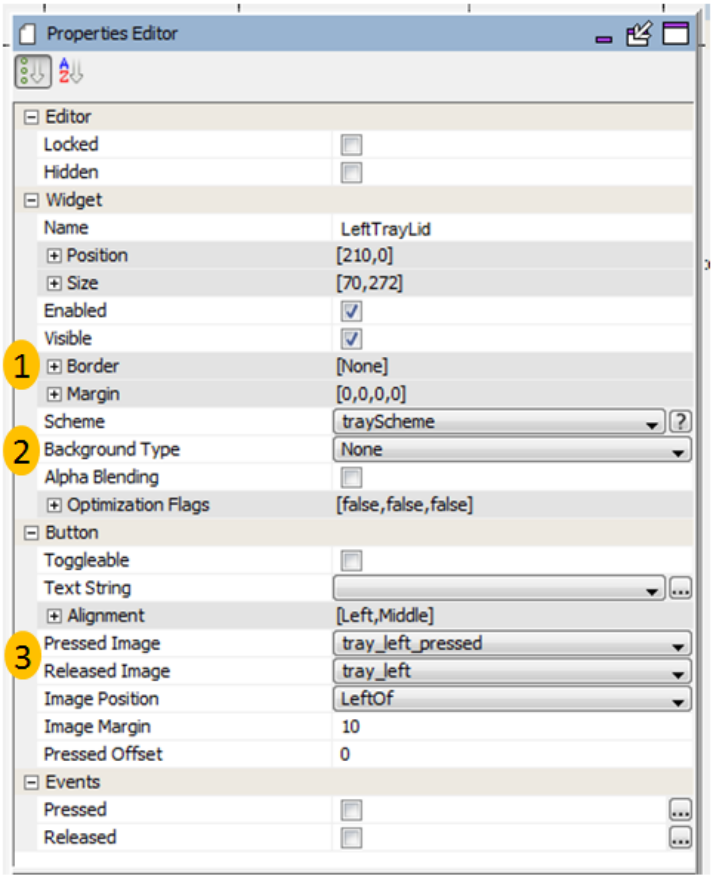


The drawing hierarchy for this part of the design is shows that ImageWidget5 is a daughter widget to the LeftTrayLid button widget.



Examining the properties of the LeftTrayLid button widget reveals more about how this works. The following figure demonstrates these three properties.

1. The *Border* is defined as *None*.
2. *Background Type* is defined as *None*.
3. The different images used will show when the button is *Pressed* or *Released*.



By setting the border and background to *None*, the button is invisible. Only by providing different images for *Released* versus *Pressed* does the user know when the button has been pressed.

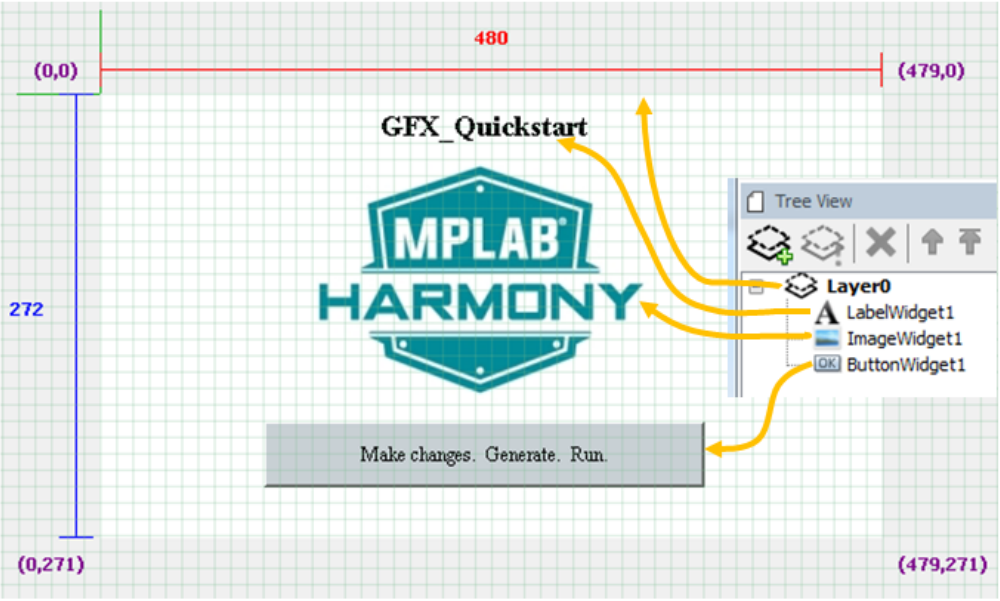
The actual touch region defined by the button is much larger than the images shown on the display. This extra area increases the touch response of the display.

Small Buttons Controlled by Phantom Buttons

Provides information on phantom button control of small buttons.

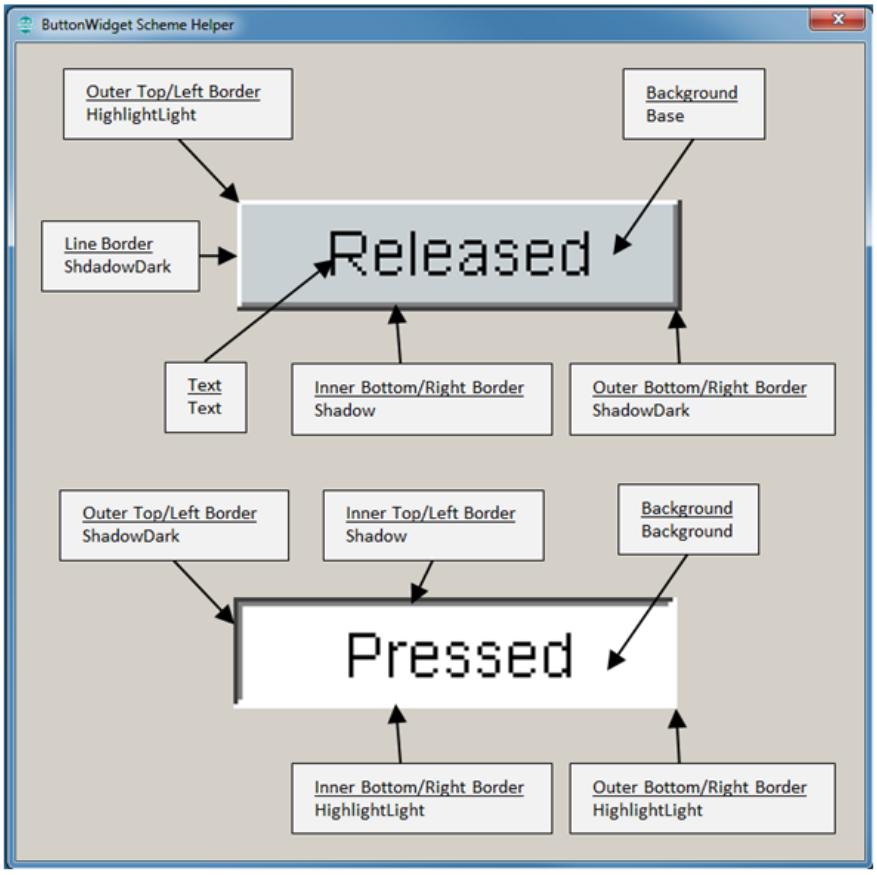
Description

When the border is not set to *None*, and the background is not set to *None*, the button widget provides a direct visible clue to the user when it is pressed. Which can be seen in the following figure with the button from aria_quickstart. In aria_quickstart, ButtonWidget1 has a bevel border, and a fill background.



Let's use aria_quickstart to demonstrate how to control ButtonWidget1 using a phantom button to surround it, thereby increasing touch responsiveness.

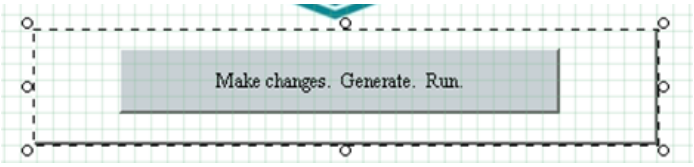
When using a bevel border and filled background, the button provides visible feedback when it is asserted.



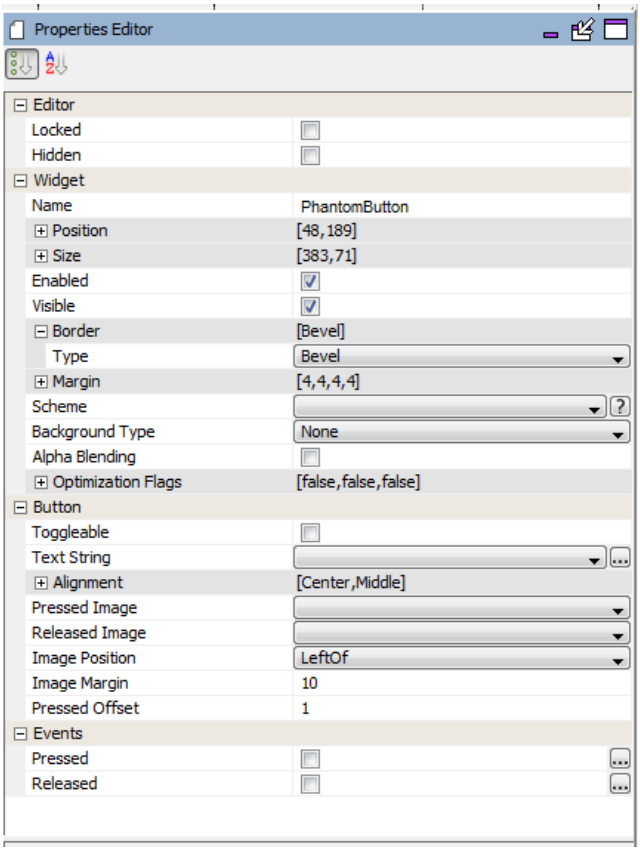
To use this feedback mechanism instead of images, there is a way to have a small button on the display, with a larger touch zone provided by another phantom button.

Steps:

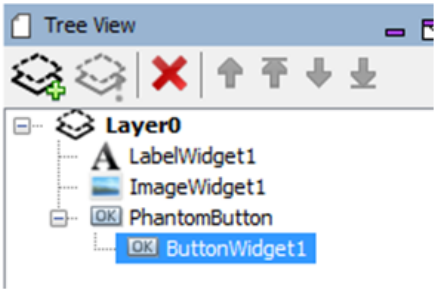
1. Click on *ButtonWidget1* in the *Screen Designer* panel. Go to the *Properties Editor* panel for the widget and uncheck the *Enabled* property to disable the button. Enable *Toggleable* so that this button will have a memory.
2. Drag a new button from the Widget Tool Box panel and center it around *ButtonWidget1*. In the *Properties Editor* panel for this new button, change the name of the widget to *PhantomButton*. Change the *Background Type* to *None*. Leave the *Border* set as *Bevel* for now. The following figure displays the new button in the *Screen Designer* panel:



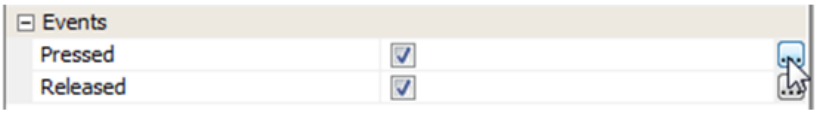
The *Properties Editor* panel should display the following information.



3. In the *Tree View* panel, drag *ButtonWidget1* to be a daughter widget of *PhantomWidget*. When *PhantomWidget* is moved, *ButtonWidget1* will move along with the parent.

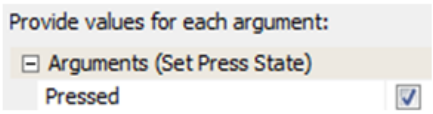


4. Click on *PhantomButton* again in the *Screen Designer panel* and move to the *Properties Editor*. Enable both the *Pressed* and *Released* events. Then click on the (…) icon to define the events. (See the following two steps.)



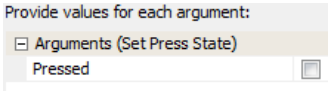
5. Defining the Pressed Event.

Click on the (…) icon. In the *Event Editor*, under *Pressed* dialog, click the *New* icon to define a new event. In the *Action Edit Dialog* that next appears, leave the selection on the template and hit the *Next* button. In the next window, select the target of the event. We want to change the state of *ButtonWidget1*, so select it and hit *Next*. The next dialog shows all the template actions that we can use to modify ButtonWidget1. Choose *Set Pressed State* and hit *Next*. Set the Argument to *Enable Pressed*. Name this event *Set Press state for ButtonWidget1* then hit Finish. Leave the Event Editor by hitting *Ok*.



6. Defining the Released Event.

Click on the (…) icon. In the *Event Editor*, under *Released dialog*, click the New icon to define a new event. In the *Action Edit Dialog* that next appears, leave the selection on the template and hit the *Next* button. In the next window, select the target of the event. We want to change the state of ButtonWidget1, so select it and hit *Next*. Choose *Set Pressed State* and hit *Next*. Leave the *Argument* disabled. Name this event *Unset Press state for ButtonWidget1* then hit Finish. Leave the Event Editor by hitting *Ok*.



7. Generate the application from the MPLAB Harmony Configurator main menu.

8. From the MPLAB main menu, build and run the project. To verify that ButtonWidget1 does change, click outside of the original boundaries.

9. As a final step, hide the PhantomButton by changing its border to *None*. Next, Generate the code again from MHC. Finally, build and run the project from MPLAB and see how much easier it is to assert ButtonWidget1 using a phantom button.

GPU Hardware Accelerated Features

This section details how to configure the GPU hardware accelerated features.

Description

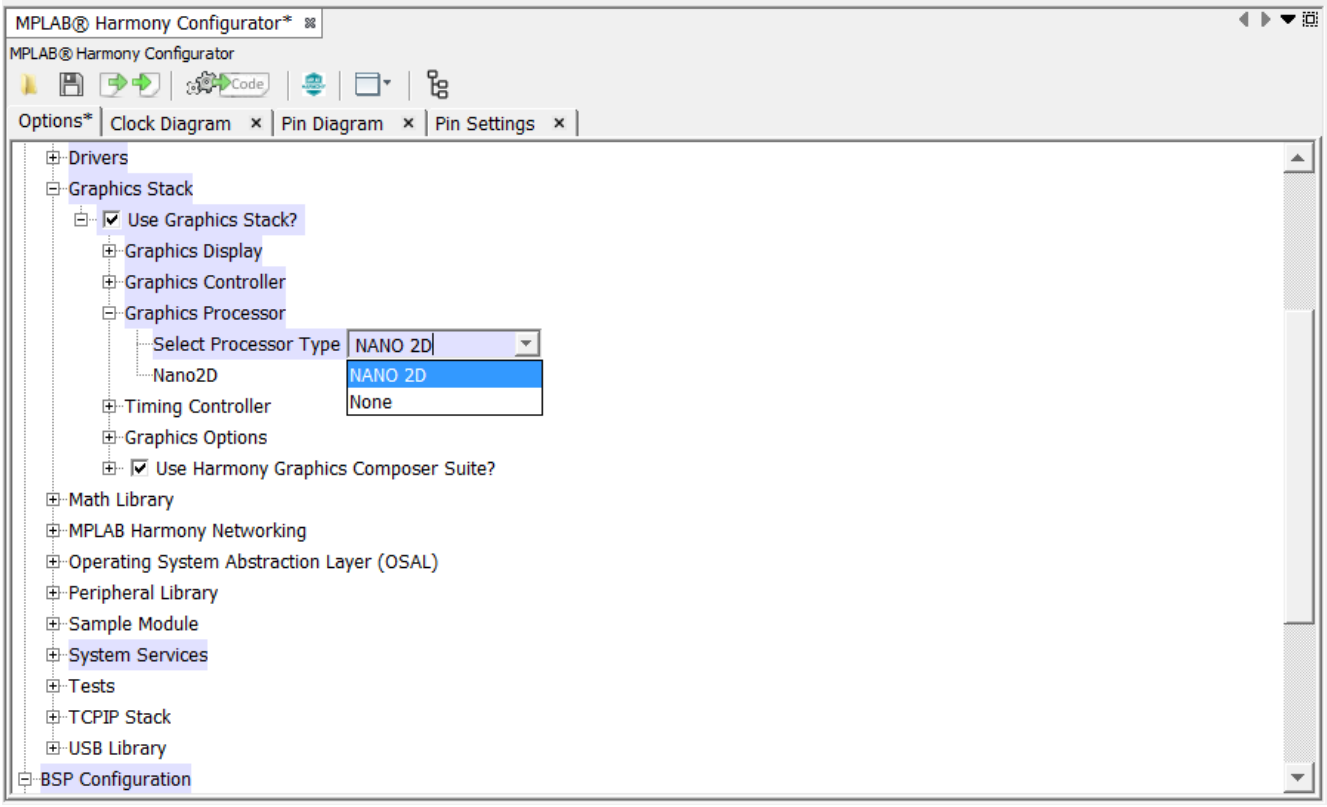
On the PIC32MZ DA devices, the on-board 2D Graphics Processing Unit (GPU) peripheral allows certain features to be accelerated. These features are:

- Line draw
- Single-color rectangle fill
- Image Blit

Once configured, these features are supported by the Hardware Abstraction Layer (HAL) and can be enabled or disabled at run-time. When disabled, the HAL falls back to the software-based algorithms, and relies on the CPU to perform the features.

Configuring for GPU Hardware Acceleration

The Nano2D library, is the driver library that permits hardware acceleration via the GPU. To make sure the Nano2D library is configured as part of your application, make sure to enable this in the MPLAB Harmony Configurator (MHC) under *Graphics Stack* > *Use Graphics Stack* > *Graphics Processor* > *Select Processor Type* > *NANO 2D*.



Enabling/Disabling GPU Hardware Acceleration at Runtime

Once configured, the hardware acceleration via the GPU is enabled by default at launch. The hardware acceleration can subsequently be turned on or off at runtime by calling the following lines of code:

Enable acceleration:

```
GFX_Set(GFXF_DRAW_PIPELINE_MODE, GFX_PIPELINE_GCUGPU);
```

Disable acceleration:

```
GFX_Set(GFXF_DRAW_PIPELINE_MODE, GFX_PIPELINE_GCU);
```

This change takes effect immediately for subsequent draw instructions into HAL.

Line Draw and Rectangle Fill Hardware Acceleration

When the GPU hardware acceleration is enabled, line draw and rectangle fill features are automatically supported. This is supported by HAL function calls `GFX_DrawLine` and `GFX_RectFill`. The actual routing of the call between the hardware accelerated support versus the software-based algorithmic support is abstracted from the caller.

The following table displays performance improvement by comparing the frame update rate of rectangular fills of varying sizes with, and without hardware acceleration. The table shows that the higher the frame update rate, the better the performance. The measurement is performed using the entire Harmony Graphics Stack but with most Aria draw pipeline features disabled, so that the focus is on HAL performance.

Rect Fill Size	No Acceleration Frame Update Frequency (Hz)	Hardware accelerated Frame Update Frequency (Hz)	Performance Improvement
60x60	101	160	58.4%
100x100	37	158	327.0%
140x140	19	157	726.3%
180x180	11	156	1318.2%
220x220	8	155	1837.5%

Note: The HAL uses a software algorithm for rectangle fill sizes below 50x50, as the CPU is able to perform the operation faster than the GPU below that size.

Image Blit Hardware Acceleration

The only way Image Blits significantly leverage hardware acceleration is via the block transfer of image data that has been preprocessed into DDR/Internal SRAM memory into frame buffer memory.

Note: The GPU is able to interpret and transfer pixel data in RGB565 or RGBA8888 format only.

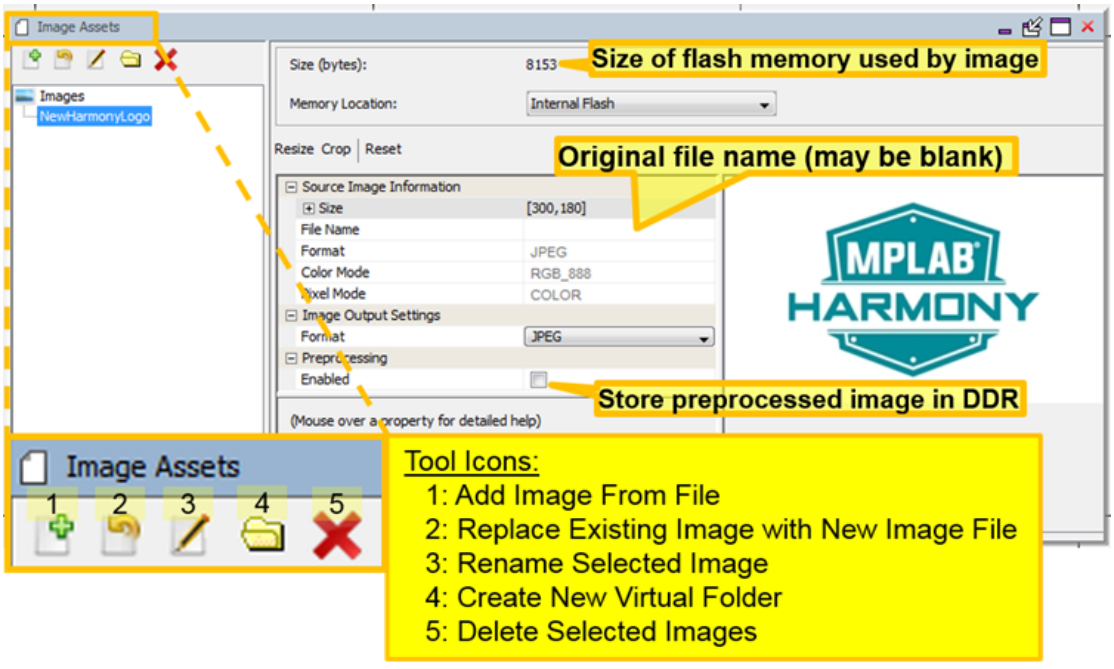
The following table displays performance improvement by comparing the frame update rate of the image blit of the same 100x100 image in varying formats with, and without GPU acceleration. The table shows that the higher the frame update rate, the better the performance. There is a marked performance increase when using the preprocessing method (despite the amount of image data is doubled in RGBA8888 versus RGB565).

Image Format (100x100)	No Acceleration Frame Update Frequency (Hz)	GPU Frame Update Frequency (Hz)	Performance Improvement
RGB565 raw pixels	37	60	62.1%
RGB565 with RLE compression	26	34	30.8%
JPEG (24-bit)	17	22	29.4%
PNG (32-bit)	13	15	15.4%
Preprocessed RGBA8888 raw pixels	29	161	455.2%

The GPU works best with image sizes in powers of two (such as 128x128 instead of 125x105). Images with sizes that are not a power of two may be rendered with artifacts. This is often a case-by-case situation and the way to remedy this is to pad the memory footprint up to the nearest power of two.

Prior to application use, images stored in flash storage will need to be preprocessed, converting them from the original format into a raw bitmap. There are two methods to achieve this:

1. Calling from application code: The API GFXU_Preprocess Image can be used to preprocess an image asset to a target memory location (DDR or internal SRAM) while specifying the destination color mode (RGB565 or RGBA8888). The application developer will need to manage the target memory and be careful not to stomp on other critical memory structures such as the frame buffer, or the GPU's command buffer. Power of two padding can be enabled via the API.
2. The application developer can also use the Image Assets options within the MPLAB Harmony Graphics Composer User's Guide (MHGC) to specify that certain image assets should be preprocessed at application launch. This can be achieved by enabling image preprocessing as shown under the Preprocessing sub-section of the Image Asset window as shown in the following figure:



For more information, see Image Assets and DDR Organizer under the Graphics Composer Asset Management section above.

Image Preprocessing Memory Management

This sections describes preprocessing.

Description

Whether using internal SRAM only or DDR memory, care must be taken when allocating memory for preprocessing images. For more information, see Image Assets and DDR Organizer under the Graphics Composer Asset Management section above.

Preprocessing using DDR

For PIC32MZ DA devices with access to DDR memory, the frame buffer and the command buffer for the GPU is also located on the DDR. It is important for the application developer to select the appropriate memory location in DDR for image preprocessing without trampling on these other memory structures.

The following table specifies the available addressing region to access the DDR memory.

Device Type	Address Range Begin (KSEG1)	Address Range End (KSEG1)
Internal DDR (maximum size 32 MB)	0xA8000000	0xA9FFFFFF
External DDR (maximum size 128 MB)	0xA8000000	0xAFFFFFFF

At configuration time, MHGC generates the frame buffer allocation in the application's system configuration code.

This allocation is targeting a WVGA RGBA8888 3-overlay double-buffered configuration; therefore, six buffer allocations are specified. More DDR memory can be freed up for image preprocessing using the following:

- WVGA Resolution is not required
- Enable all three overlays
- Double frame buffering

The application developer may choose to change the allocation manually in `system_config.h`.

The following table breaks down the allocation:

Frame Buffer	Address Range Begin	Address Range End
Layer0 Buffer 0	0xA8000000	0xA8176FFF
Layer0 Buffer 1	0xA8465000	0xA85DBFFF
Layer1 Buffer 0	0xA8177000	0xA82EDFFF
Layer1 Buffer 1	0xA85DC000	0xA8752FFF
Layer2 Buffer0	0xA82EE000	0xA8464FFF
Layer2 Buffer1	0xA8753000	0xA88CBFFF

For an example on using image preprocessing using DDR memory, please refer to the aria_coffee_maker application.

Internal SRAM Only

When operating with only the internal SRAM, the frame buffer can take up a significant portion of available memory. To avoid system stability issues with dynamically allocating memory for the preprocessing, the application developer may want to predetermine the memory footprint required for the image and assign the memory statically.

For an example of image preprocessing using internal SRAM, please refer to the aria_radial_menu application.

Creating a MPLAB Harmony Graphics Application Using a Third-Party Display

This demonstration provides a step-by-step example of how to create a MPLAB Harmony graphics application using a non-Microchip (third-party) display.

Description

Introduction

Creating a new MPLAB Harmony graphics application using a Microchip board and a Microchip display is very simple: A new MPLAB Harmony application is created and the Board Support Package (BSP) belonging to the hardware configuration is selected. If the project is using a third-party display then there are more steps and this tutorial will provide an example of the process.

This tutorial shows how to connect a third-party display to the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit board (EF Starter Kit) using two Microchip Adapter boards and a custom ribbon cable. It shows how to setup the pinouts, configure graphics, and adapt an existing MPLAB Harmony capacitive touch driver to support the display board's capacitive touch controller.

Prerequisites

Before beginning this tutorial, ensure that the MPLAB X IDE is installed along with the necessary language tools as described in Volume I: Getting Started With MPLAB Harmony > Prerequisites. In addition, ensure that MPLAB Harmony is installed on the hard drive, and that the correct MPLAB Harmony Configurator (MHC) plug-in is installed in the MPLAB X IDE.

A basic familiarity with application development under MPLAB X and MPLAB Harmony is required, including how to use MPLAB Harmony Configurator (MHC). There are introductory videos on Microchip's YouTube channel for those who have never used MPLAB Harmony. The first video to watch is Getting Started with MPLAB Harmony. There is also a Creating Your First Project tutorial in Volume 1 of MPLAB Harmony's documentation.

For first time users of MPLAB Harmony Graphics there is a video series on YouTube. The first video is MM MPLAB® Harmony Edition - Ep. 7 - MPLAB Harmony Graphics Composer Suite. In Volume 1 of MPLAB Harmony's documentation there are Quick Start tutorials covering graphics, located at Quick Start Guides > Graphics and Touch Quick Start Guides.

Tutorial Resources

The folder `./apps/examples` in MPLAB Harmony has a project that can be copied and used as the base of this tutorial, `3rd_party_display_start`, and a project that represents the completed project from this tutorial, `3rd_party_display.`

This is what you will find in the `./apps/examples` folder under Harmony 2.06:

3rd_party_display

3rd_party_display_start

creating_your_first_project

peripheral

events_testbed

system

If there are difficulties then compare the completed project with the current project.

Tutorial Hardware

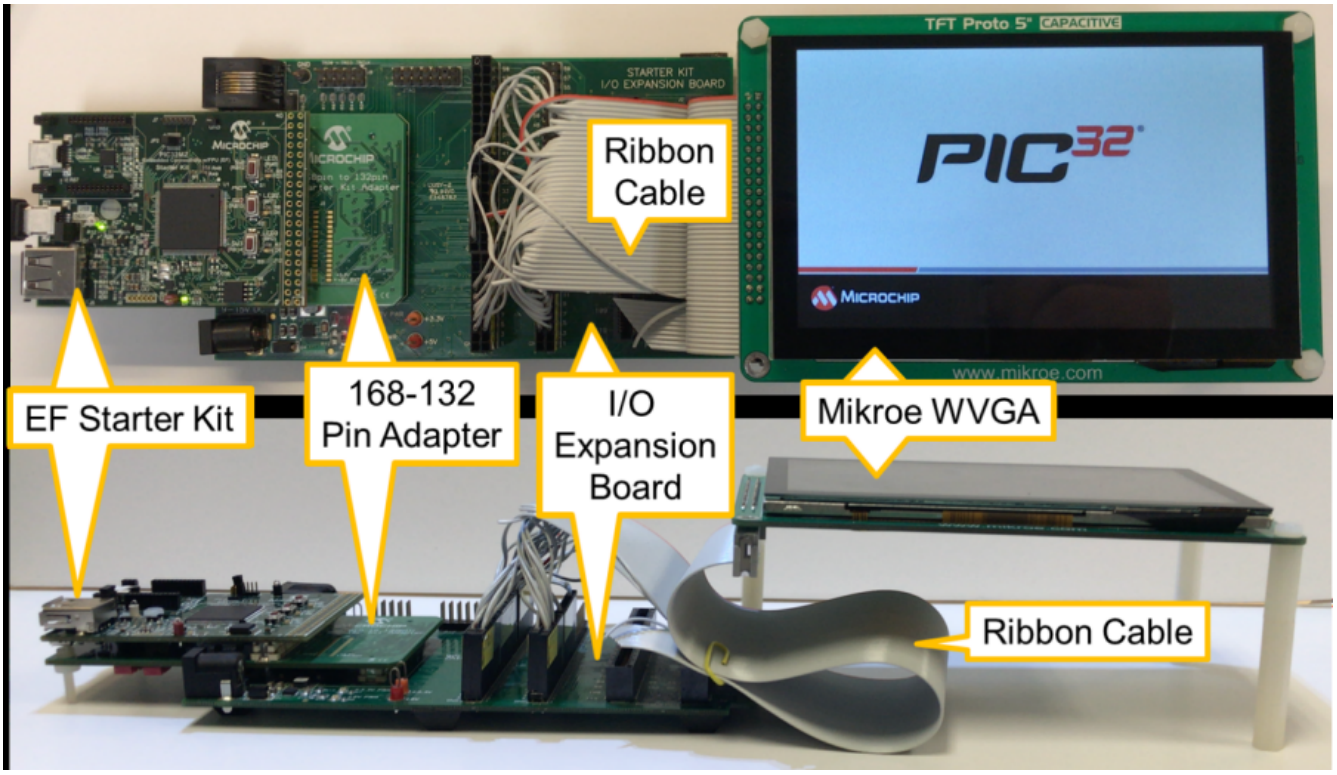
Of all the PIC32MZ devices available today, the PIC32MZ EF family is the best candidate for this effort. The EF family does not have on-chip graphics controller or Graphics Processing Unit (GPU), which makes it a less expensive and lower power solution for use with a display that has a built-in controller.

Mikroelektronika (Mikroe) offers a prototype display that can be used using a ribbon cable between the display and the EF host. This third party (non-Microchip) board serves as the basis for this tutorial. The 'TFT PROTO 5" Capacitive' display costs around 100USD and is available for order online (https://www.mikroe.com/tft-proto-5-capacitive-board). It has an 800x480 pixel WVGA display, driven by an SSD1963 graphics controller. The SSD1963 graphics controller is already supported in MPLAB Harmony. It has a Focal Tech FT5x06 capacitive touch controller. This tutorial will cover how to design the pin-out between the EF host and display board, as well as how to adapt an existing MPLAB Harmony capacitive touch driver (MTCH6303) to support the Focal Tech touch controller.

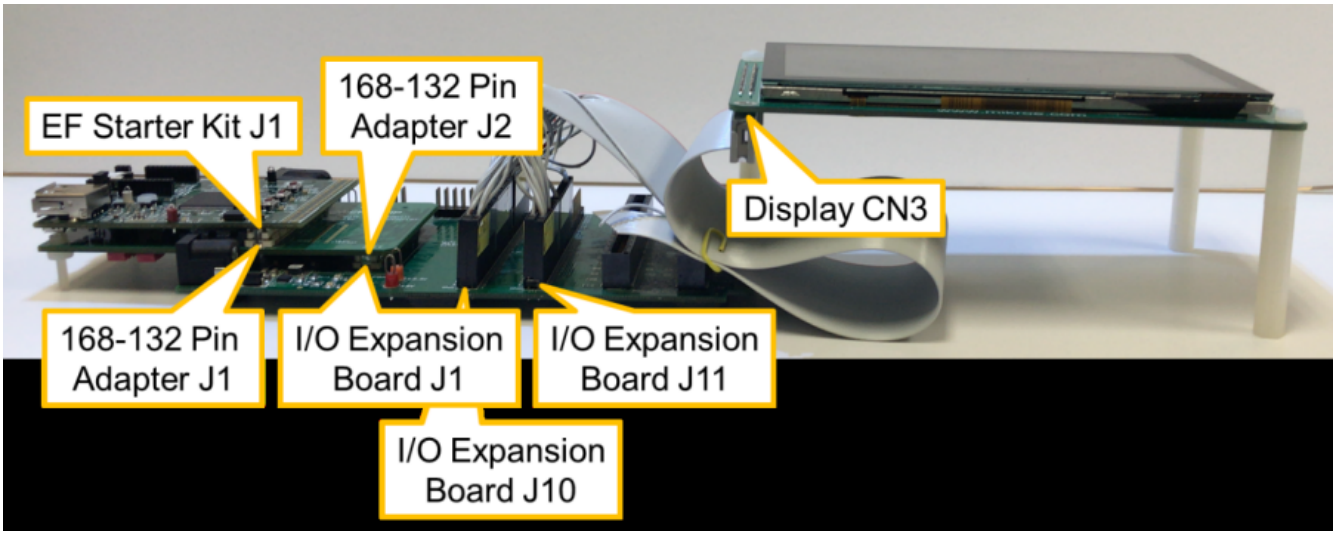
For this tutorial the following hardware will be used:

1. PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit board (Part # DM320007).
2. Starter Kit I/O Expansion Board (Part # DM320002) – this provides the 0.1" headers we need to connect up the display using a ribbon cable or 0.1" jumpers.
3. PIC32MZ Starter Kit Adaptor Board (Part # AC320006) – this provides an 168 to 132 pin adapter to adapt the 168-pin connector on the EF starter kit with the 132 pin connector on the I/O Expansion Board.
4. Mikroelektronika TFT PROTO 5" Capacitive display.
5. 40 to 50 pin ribbon cable to connect the I/O Expansion Board to the display, or a set of colored 0.1" jumpers.

Here is how the hardware is assembled:



The connectors that route signals from the EF pins to the display's ribbon cable are:



The EF Starter Kit + 168-132 Pin Adapter + I/0 Expansion board can host any number of prototype hardware configurations. A spreadsheet has been developed that maps every pin of the EF device to a pin on the I/O Expansion board, with one final spreadsheet tab that provides the pin outs for the ribbon cable that connects the display to the I/O Expansion Board. (The spreadsheet is found in the Zip file `.\apps\examples\3rd_party_display\pinouts.zip`.) The picture above shows the board connectors used in getting from a pin on the EF device to a pin on the display's ribbon connector.

This spreadsheet has the following tabs:

1. **Sorted by Skit J1 Pins** – This tab maps EF pins to pins on the J1 (168 pin) connector on the 168-132 pin adapter. It also maps the 168-pin J1 connecter to the J2 132-pin connecter. Pins are sorted by the pin order on the Starter Kit 168-pin J1 connector.
2. **Sorted by Device Pins** – A copy of the first tab, sorted by EF device pins.
3. **Sorted by Adaptor J2 Pins** – A copy of the first tab, sorted by the pins on the J2 132-pin adaptor.
4. **PIC32 IO Expansion Pin Out** – Provides the pin out of the I/O Expansion Board from the 132-pin J1 connector to the 0.1" pitch headers on the board (J10,J11).
5. **End to End** – maps the EF device pins to the 0.1" pitch headers on the I/O Expansion Board. This tab can be reused to map out other application pin outs.
6. **Mikroe Display** – Provides the pin outs for the 40-pin ribbon cable connector (CN3) on the display board.

7. **End to End by Device Pins** – This tab combines Tab 4 with Tab 6. It shows how to build a ribbon cable between the I/O Expansion Board and the display. On this tab the rows belonging to EF device pins that aren't part of the ribbon cable are hidden for the sake of simplicity.

Tab 7 of the spreadsheet shows:

Dev Pin #	Device Pin Name	J2 Pin #	J2 Pin Name	J1 Pin #	J1 Pin Name	J10 Pin #	J11 Pin #	Pin #	Pin Name		Notes:
3	EBID5/AN17/RPE5/PMD5/RE5	13	EBID5/PMD5	13	PMPD5/RE5	7	—	18	TFT-D5	p	
4	EBID6/AN16/PMD6/RE6	9	EBID6/PMD6	9	PMPD6/RE6	6	—	19	TFT-D6	p	
5	EBID7/AN15/PMD7/RE7	7	EBID7/PMD7	7	PMPD7/RE7	5	—	20	TFT-D7	p	
12	EBIWE/AN20/RPC3/PMWR/RC3	28	CAN1_RX via JP1 Open, JP2 closed	28	PMPWR/RD4	14	—	10	TFT-WR#	g	Bit banged as part of every command or data sequence, also used in device reset
13	EBIOE/AN19/RPC4/PMRD/RC4	25	EBIOE/PMRD/RC4	25	PMPRD/RD5	13	—	11	TFT-RD#	g	Bit banged as part of device reset
22	TMS/AN24/RA0	126	TMS/RA0	126	TMS/RA0	—	60	7	TFT-RST#	g	To pin 127 of SSD1963 (RESET#), GND via 10K Ohm, RESET_BAR
87	EBIA14/PMCS1/PMA14/RA4	27	EBIA14	27	PMPCS1/RD11	15	—	9	TFT-CS#	g	Chip Select is bit banged as part of every code sequence
95	RPA14/SCL1/RA14	84	RPA14/SCL1/RA14	84	SCL1	—	35	3	CTP-SCL	I	I2C interface to touch controller
96	RPA15/SDA1/RA15	86	SDA1/TOUCH_SDA	86	SDA1	—	37	4	CTP-SDA	I	I2C interface to touch controller
97	EBIA15/RPD9/PMCS2/PMA15/RD9	79	SS3/EBIA15	79	INT4	36	—	38	CTP-RST#	g	Pulled high on Mikroe board by J2A
98	RPD10/SCK4/RD10	95	SCK4/WIFI_SCK/SD13	95	SD01	43	—	39	CTP-WAKE#	g	Pulled high on Mikroe board by J1A
104	RPD0/RTCC/INT0/RD0	87	INT0/RD0	87	INT0	42	—	5	CTP-INT#	g	Can be sent to INT*n*
109	RPD1/SCK1/RD1	91	RD1/SCK1/AUDIO_BICK	91	SCK1	41	—	8	TFT-D/C#	g	Data/Command_bar for SSD1963
110	EBID14/RPD2/PMD14/RD2	24	EBID14/PMD14	24	PMPD14/RD6	—	12	27	TFT-D14	p	
111	EBID15/RPD3/PMD15/RD3	26	EBID15/PMD15	26	PMPD15/RD7	—	11	28	TFT-D15	p	
112	EBID12/RPD12/PMD12/RD12	20	EBID12/PMD12	20	PMPD12/RD12	—	10	25	TFT-D12	p	
113	EBID13/PMD13/RD13	22	EBID13/PMD13	22	PMPD13/RD13	—	9	26	TFT-D13	p	
121	ETXCLK/RPD7/RD7	47	RD7/U2RTS/BT_RTS	47	SDI2	24	—	12	TFT-TE	g	Wired to pin 50 of SSD1963 (TE, Tear Enable) to be wired to GPIO on EF, Optional
124	EBID11/RPF0/PMD11/RF0	18	PMD11 via JP1, JP2 Open	18	PMPD11/RF0	—	7	24	TFT-D11	p	
125	EBID10/RPF1/PMD10/RF1	16	PMD10 via JP3 with JP4 open	16	PMPD10/RF1	—	8	23	TFT-D10	p	
127	EBID9/RPG1/PMD9/RG1	14	EBID9/PMD9	14	PMPD9/RG1	—	5	22	TFT-D9	p	
128	EBID8/RPG0/PMD8/RG0	10	EBID8/PMD8	10	PMPD8/RG0	—	6	21	TFT-D8	p	
135	EBID0/PMD0/RE0	23	EBID0/PMD0	23	PMPD0/RE0	12	—	13	TFT-D0	p	
138	EBID1/PMD1/RE1	21	EBID1/PMD1	21	PMPD1/RE1	11	—	14	TFT-D1	p	
142	EBID2/PMD2/RE2	19	EBID2/PMD2	19	PMPD2/RE2	10	—	15	TFT-D2	p	
143	EBID3/RPE3/PMD3/RE3	17	EBID3/PMD3	17	PMPD3/RE3	9	—	16	TFT-D3	p	
144	EBID4/AN18/PMD4/RE4	15	EBID4/PMD4	15	PMPD4/RE4	8	—	17	TFT-D4	p	
				11	GND	2, 22, 39	2, 22, 39	2	GND		

The ribbon cable for this project is constructed using the map from J10 Pin#/J11 Pin # to the TFT Proto 5" Pin #. For example, the first line of the Tab 7 shows that pin 7 of the J10 header on the I/O expansion board is connected to pin 18 of the display connector, thereby connecting PMPD5 (PMP data pin 5) on the device to TFT-D5 on the display.

Note: display pins with a "#" suffix indicate that the signal is active low (# = bar).

TFT-Dn display pins are part of the SSD1963 display controller's Parallel Master Port (PMP) interface. Other TFT-* pins are part of the controller to host interface. For example, TFT-WR# is connected to the controller's WRbar (write strobe bar) pin, which is called WR_STROBE_BAR in the MPLAB Harmony Graphical Pin Manager. (Setting up the project's pins using the Pin Manager is discussed later in the tutorial.)

On the display connector FT5x06 capacitive touch controller pins are called CTP-*. There is an I2C clock pin (CTP-SCL), I2C data pin (CTP-SDA), an interrupt pin to alert the host of a touch event (CTP-INT#), and reset/wakeup pins (CTP-RST#/CTP-WAKE#).

Creating the Project in MPLAB and MPLAB Harmony

Getting Started

The pre-installed project, 3rd_party_display_start can be used as a basis for the work discussed in this tutorial. Be sure to copy this project to a place in the MPLAB Harmony directory hierarchy that is just as deep. If this is not done, all the relative paths in the project's configuration will no longer find the project's files and nothing will build.

For example, copying `3rd_party_display_start` into a directory `.\apps\3rd_party_display` will not work, since the target directory is one level higher in MPLAB Harmony's directory hierarchy. The directory `.\apps\gfx\3rd_party_display` will work since it is at the same level in the hierarchy.

There is an extra file in the `.\apps\examples\3rd_party_display_start` file (`xc32_vm.nn_pic32mx_include_assert.h`) , which provides the modification to the compiler's `assert.h` as discussed in Volume 1 of MPLAB Harmony's documentation (Creating Your First Project). This modification supports producing breakpoints under the debugger when an assert fails, which can be very useful in debugging the code. Simply use this file to replace `./xc32/vm.nn/pic32mx/include/assert.h,where m.nn represents the version number of the compiler you are using.`

For first time users of the PIC32MZ product line and MPLAB Harmony should create the starting the project from scratch. Follow the instructions in "Creating Your First Project", which is found in Volume 1: Getting Started With MPLAB Harmony Libraries and Applications. Call the new project *`3rdPartyDisplay`* instead of *Heartbeat*.

In Part 1, Step 3 of the Creating Your First Project, use a different application name than "heartbeat." For example accept the default "app", then replace "heartbeat" with the new application name in the tutorial code examples. If the default application name "app" is used then "heartbeat" is replaced by "app" in the code examples. The header file heartbeat.h would be named `app.h` instead and it should contain:

```
typedef enum
{
/* Application's state machine's initial state. */
APP_STATE_INIT=0,
APP_STATE_SERVICE_TASKS,

/* TODO: Define states used by the application state machine. */
APP_RESTART_TIMER

} APP_STATES;
```

Here the enum is called APP_STATES instead of HEARTBEAT_STATES and the state APP_RESTART_TIMER replaces the state HEARTBEAT_RESTART_TIMER. The structure HEARTBEAT_DATA is now called APP_DATA:

```
typedef struct
{
/* The application's current state */
APP_STATES state;

/* TODO: Define any additional data used by the application. */
SYS_TMR_HANDLE hDelayTimer; // Handle for delay timer

} APP_DATA;
```

The same principle applies to app.c (instead of heartbeat.c in the tutorial). The structure heartbeatData is now called appData. The source file app.c should contain:

```
{

/* Check the application's current state. */
switch ( appData.state )
{
/* Application's initial state. */
case APP_STATE_INIT:
{
bool appInitialized = true;



if (appInitialized)
{
appData.hDelayTimer = SYS_TMR_DelayMS(HEARTBEAT_DELAY);
if (appData.hDelayTimer != SYS_TMR_HANDLE_INVALID)
{ // Valid handle returned
BSP_LEDOn(HEARTBEAT_LED);
appData.state = APP_STATE_SERVICE_TASKS;
}
appData.state = APP_STATE_SERVICE_TASKS;
}
break;
}

case APP_STATE_SERVICE_TASKS:
{
if (SYS_TMR_DelayStatusGet(appData.hDelayTimer))
{ // Single shot timer has now timed out.
BSP_LEDToggle(HEARTBEAT_LED);
appData.state = APP_RESTART_TIMER;
}
break;

}

/* TODO: implement your application state machine.*/
case APP_RESTART_TIMER:
{ // Create a new timer
appData.hDelayTimer = SYS_TMR_DelayMS(HEARTBEAT_DELAY);
if (appData.hDelayTimer != SYS_TMR_HANDLE_INVALID)
{ // Valid handle returned
appData.state = APP_STATE_SERVICE_TASKS;
}
```

```
break;
}

/* The default state should never be executed. */
default:
{
/* TODO: Handle error in application's state machine. */
break;
}
}
}
```

At the end of the *Creating Your First Project* tutorial, the project supports a HyperTerminal console on a PC, which can be used to display diagnostic messages. The project will also support the advanced error handling (asserts and exceptions) that MPLAB Harmony provides.

When running this application, verify that the HyperTerminal application (115200 baud, 8 bits, no stop bits) sees an initialization message of, *Application created Mar 1 2018 15:09:50 initialized!* at startup, where the date and time report when the `app.c` file was last compiled. This message originates in the application initialization function:

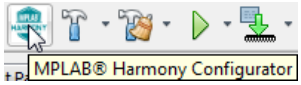
```
void APP_Initialize ( void )
{
SYS_MESSAGE("\r\nApplication created " __DATE__ " " __TIME__ " initialized!\r\n");

//Test out error handling
// assert(0);
// {
// uint8_t x, y, z;
// x = 1;
// y = 0;
// z = x/y;
// SYS_DEBUG_PRINT(SYS_ERROR_DEBUG,"x: %d, y: %d, z: %d\r\n",x,y,z);
// }
/* Place the App state machine in its initial state. */
appData.state = APP_STATE_INIT;


/* TODO: Initialize your application's state machine and other
* parameters.
*/
}
```

Verify that asserts and exception handling work before proceeding. Uncomment the assert and test. Then comment out the assert and uncomment the {…} clause to test out exceptions.

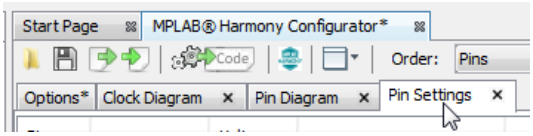
Note: If this is the first time hooking up a HyperTerminal session to the EF Starter Kit using the MCP2221, see Part 3 of the Creating Your First Project tutorial in Volume 1 of MPLAB Harmony's documentation. This part of the tutorial shows how to hookup the EF Starter Kit to your PC. It also discusses in Steps 11 and 12 how to setup your HyperTerminal application.

Setting Up Pins using the MPLAB Harmony Graphical Pin Manager

Since a pre-defined Board Support Package is not available, pin assignments will have to be manually entered into the Pin Manger using the "Pin Settings" tab. Load the startup project, either from a copy made from `.\apps\examples\3rd_party_display_start` or one created from scratch. Then run MPLAB Harmony Configurator:



From MPLAB Harmony Configurator, select the Pin Settings tab:

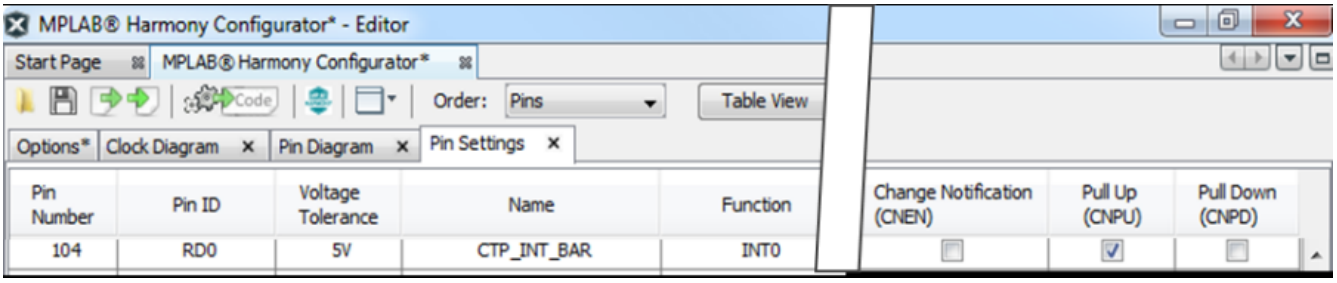


Make these modifications to the pin table:

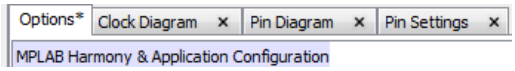
Pin Number	Pin ID	Voltage Tolerance	Name	Function	Direction (TRIS)	Latch (LAT)
1	RG15			Available	In	n/a
2	RA5			Available	In	n/a
3	RE5		PMD5	PMD5	n/a	n/a
4	RE6		PMD6	PMD6	n/a	n/a
5	RE7		PMD7	PMD7	n/a	n/a
12	RC3		WR_STROBE_BAR	GPIO_OUT	Out	Low
13	RC4		RD_STROBE_BAR	GPIO_OUT	Out	Low
14	RG6		USART to USB Bridge (BSP)	U2RX	n/a	n/a
22	RA0		RESET_BAR	GPIO_OUT	Out	Low
25	RB5		USB_VBUS_SWITCH	VBUS	Out	Low
43	RH0		BSP_LED_1	LED_AH	Out	Low
44	RH1		BSP_LED_2	LED_AH	Out	Low
45	RH2	5V	BSP_LED_3	LED_AH	Out	Low
59	RB12		BSP_SWITCH_1	SWITCH	In	Low
60	RB13		BSP_SWITCH_2	SWITCH	In	Low
61	RB14		USART to USB Bridge (BSP)	U2TX	n/a	n/a
87	RA4	5V	CHIP_SELECT_BAR	GPIO_OUT	Out	Low
95	RA14	5V	SCL1	SCL1	n/a	n/a
96	RA15	5V	SDA1	SDA1	n/a	n/a
97	RD9	5V	CTP_RST	GPIO_OUT	Out	Low
104	RD0	5V	CTP_INT_BAR	INT0	n/a	n/a
109	RD1	5V	DATA_OR_COMMAND_BAR	GPIO_OUT	Out	Low
110	RD2	5V	PMD14	PMD14	n/a	n/a
111	RD3	5V	PMD15	PMD15	n/a	n/a
112	RD12	5V	PMD12	PMD12	n/a	n/a
113	RD13	5V	PMD13	PMD13	n/a	n/a
124	RF0	5V	PMD11	PMD11	n/a	n/a
125	RF1	5V	PMD10	PMD10	n/a	n/a
126	RK7	5V		Available	In	n/a
127	RG1	5V	PMD9	PMD9	n/a	n/a
135	RE0	5V	PMD0	PMD0	n/a	n/a
138	RE1	5V	PMD1	PMD1	n/a	n/a
142	RE2	5V	PMD2	PMD2	n/a	n/a
143	RE3	5V	PMD3	PMD3	n/a	n/a
144	RE4		PMD4	PMD4	n/a	n/a

The pins labeled USART to USB Bridge (BSP) support the MCP2221 USART to USB device on the EF Starter Kit board. It provides a HyperTerminal interface on the PC. This is setup in the `3rd_party_diaplay_start` project.

Be sure the touch interrupt event interrupt (pin 104, CTP_INT_BAR) pin is pulled high (CNPU enabled), otherwise touch event interrupts will never fire:



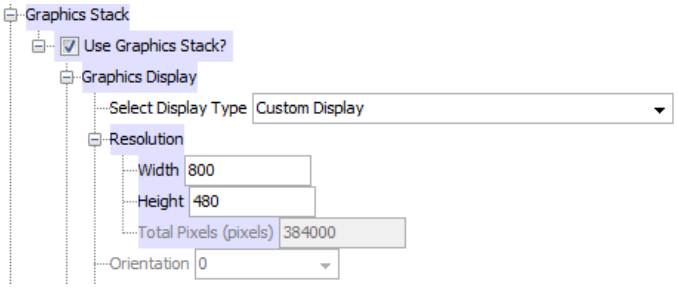
Setting up Graphics

In MPLAB Harmony Configurator, under the Options tab: 

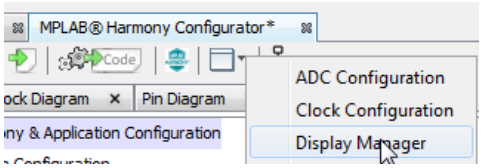
open *Harmony Framework Configuration* > *Graphics Stack* and enable the Graphics Stack with the following settings.

First select a “Custom Display” as the display type.

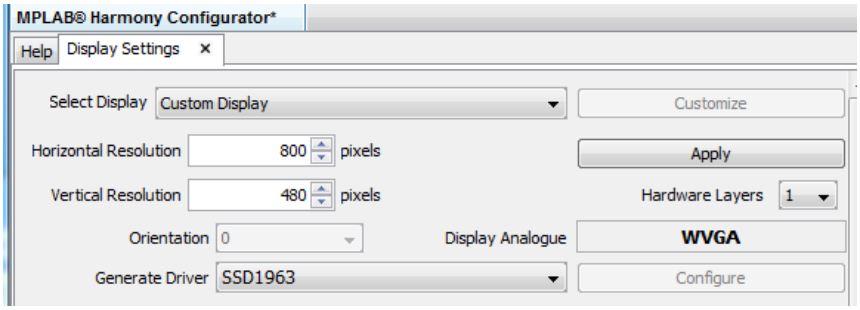
Then enter the dimensions of the Mikroe display (800x480).



Note: The display can be set in MHC’s Display Manager.

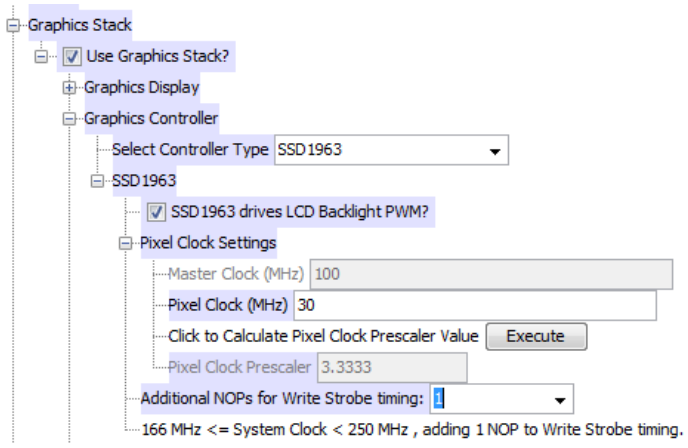


Enable the Graphics Stack using the MHC’s Options tab, it is easier to do the basic display setup here. Later the Display Manager will be used to tune the display’s timing (syncs plus front porches and back porches) so that all 800x480 pixels are correctly displayed. For now, accept the default display timings.The equivalent setup using the Display Manager is:

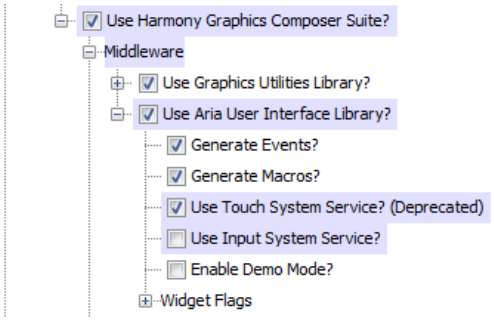


The Mikroe display uses a SSD1963 graphics controller to run the TFT display, which is supported in MPLAB Harmony. This graphics controller is connected to the EF host using the Parallel Master Port (PMP), I2C, and GPIO peripherals. (For details, see the Setting Up Pins using the MPLAB Harmony Graphical Pin Manager section above.)

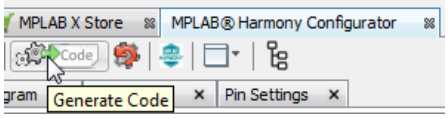
Under *Graphics Stack* > *Graphics Controller*, select the SSD1963 graphics controller, enable the controller’s backlight PWM. Change the pixel clock from the default to 30 MHz and click “Execute” to compute the Pixel Clock Prescaler value. Finally, since the system clock for the EF host runs at 200 MHz, add an additional NOP for correct Write Strobe timing.



Finally, verify *Use Touch System Service?* (Deprecated) is enabled:



When finished, re-generate the code to capture these new settings using the *Generate Code* button in MPLAB Harmony Configurator.



Be sure to use the Prompt Merge For All Differences merge strategy to maintain code customizations installed outside of MHC.

After regenerating the project, you will have to customize the `system_init.c` file, found in the project under `Source Files / app / system_config / <target_configuration>`, where `<target_configuration>` is typically "default". Move the `SYS_PORTS_Initialize` call from the middle of `SYS_Initialize` to between `SYS_DEVCON_PerformanceConfig` and `BSP_Initialize`.

The Old location:

```
  /* Initialize System Services */
// CUSTOM CODE - DO NOT DELETE
//SYS_PORTS_Initialize();
// END OF CUSTOM CODE
  sysObj.sysConsole0 = SYS_CONSOLE_Initialize(SYS_CONSOLE_INDEX_0, NULL);
```

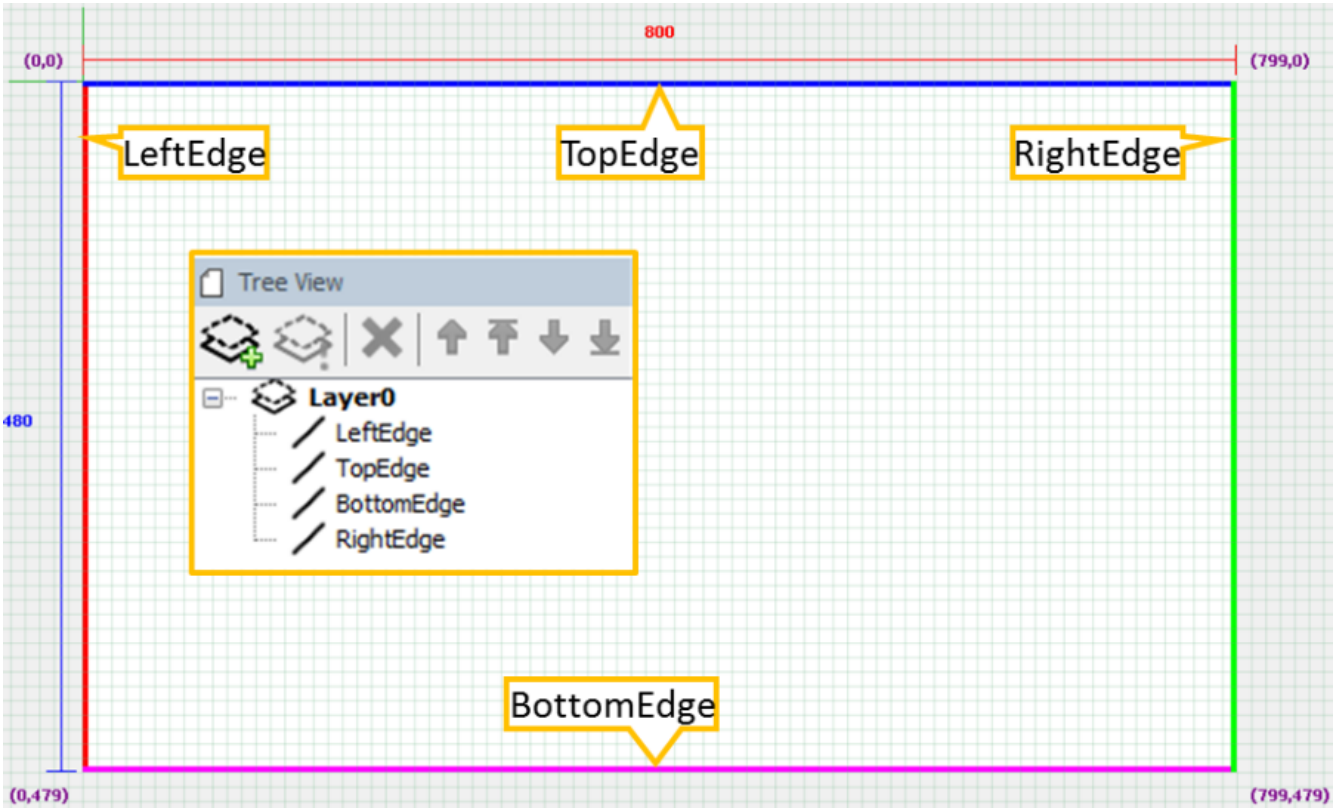
The New location is:

```
void SYS_Initialize ( void* data )
{
    /* Core Processor Initialization */
    SYS_CLK_Initialize( NULL );
    SYS_DEVCON_Initialize(SYS_DEVCON_INDEX_0, (SYS_MODULE_INIT*)NULL);
    SYS_DEVCON_PerformanceConfig(SYS_CLK_SystemFrequencyGet());
  // CUSTOM CODE - DO NOT DELETE
    SYS_PORTS_Initialize();
  // END OF CUSTOM CODE

    /* Board Support Package Initialization */
    BSP_Initialize();
```

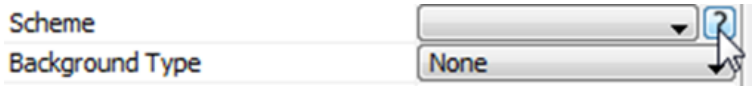
Tuning Display Timing Using Display Manager

The next step is to tune the timing of the display using the Display Manger to prevent the edges of the screen from being clipped. A rectangle needs to be drawn on the edges of the screen. Then by building and running the application, we can see if any parts of the border rectangle are

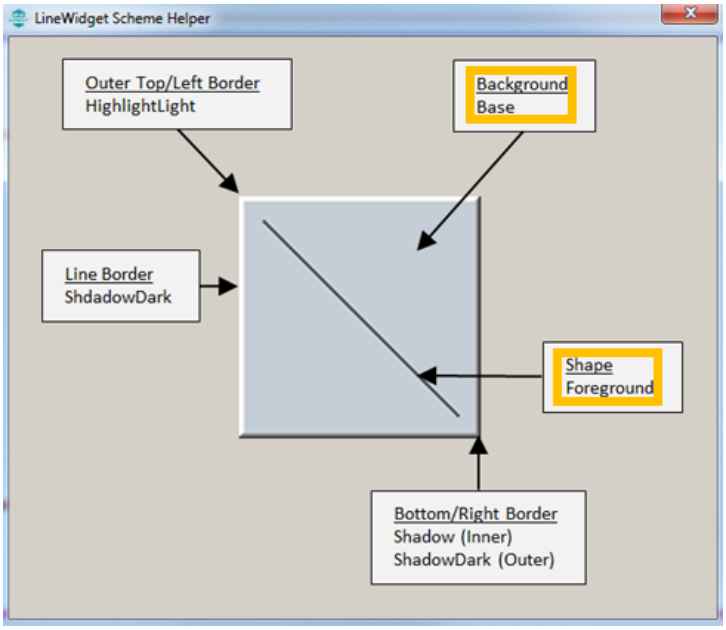
clipped or missing. A different color is needed for each of the four sides of the border rectangle, as in some cases the display controller's memory pointers can "wrap" a pixel from one side of the display to the opposite side. If all the sides are the same color this would not be apparent. Here is the screen to implement in the Screen Designer panel:



Each side of the border will require a custom color scheme. The border is created by drawing four separate lines using four separate line widgets. Examine how line widgets are colored by dragging a line widget from the Widget Toolbox panel onto the Screen Designer panel and then pick the Properties Editor Panel for that widget. Click on the "?" to the right of the Scheme property.

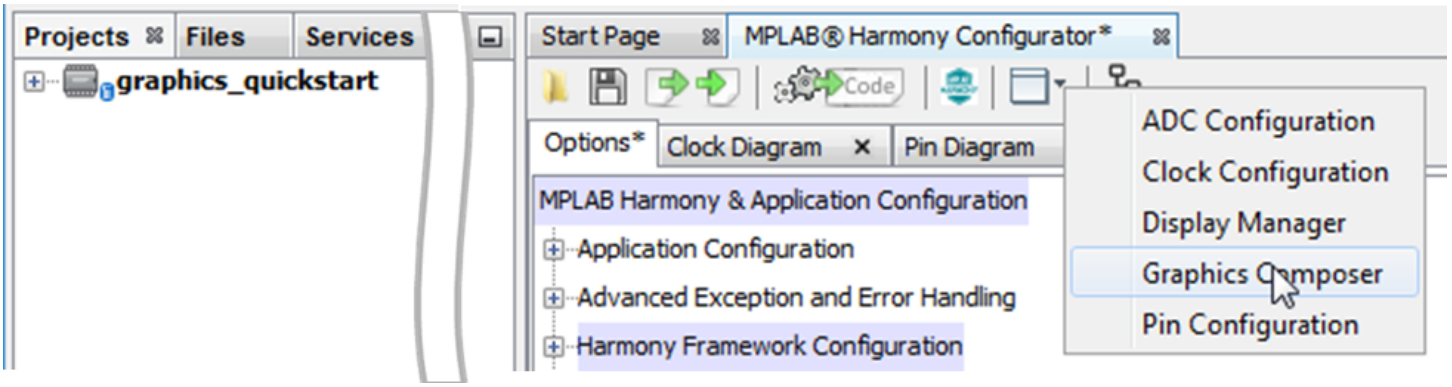


This will bring up the "Line Widget Scheme Helper" window:

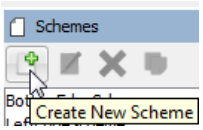


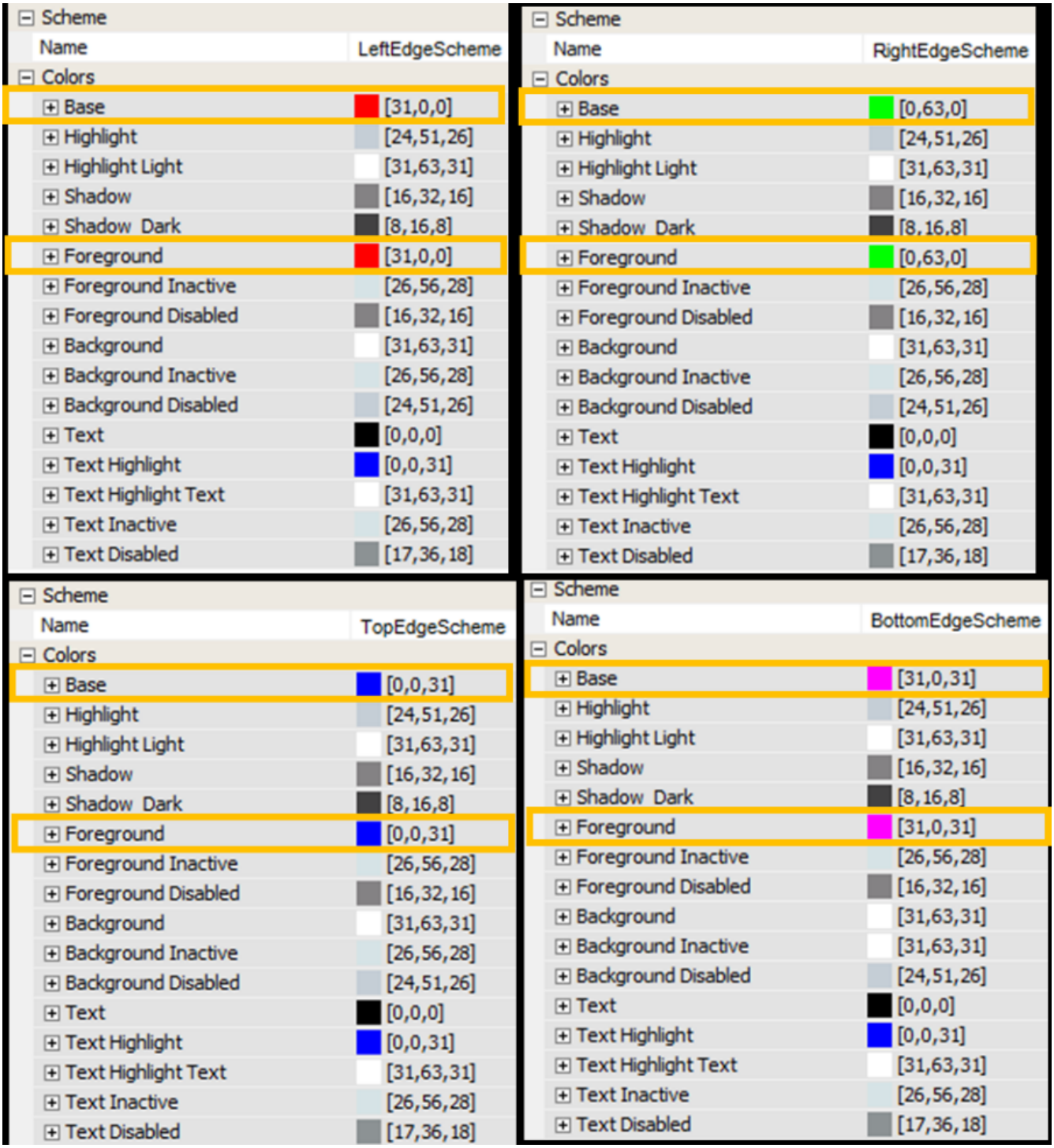
If the Background and Shape of the widget are colored with the same color, different for each side, then the four edges of the display are easily marked. Using the same colors for the line, and the widget's background, allows the use of the size and position of the line widget rather than the line's coordinates to mark that edge of the display.

To create the display, within MHC, launch the Graphics Composer.

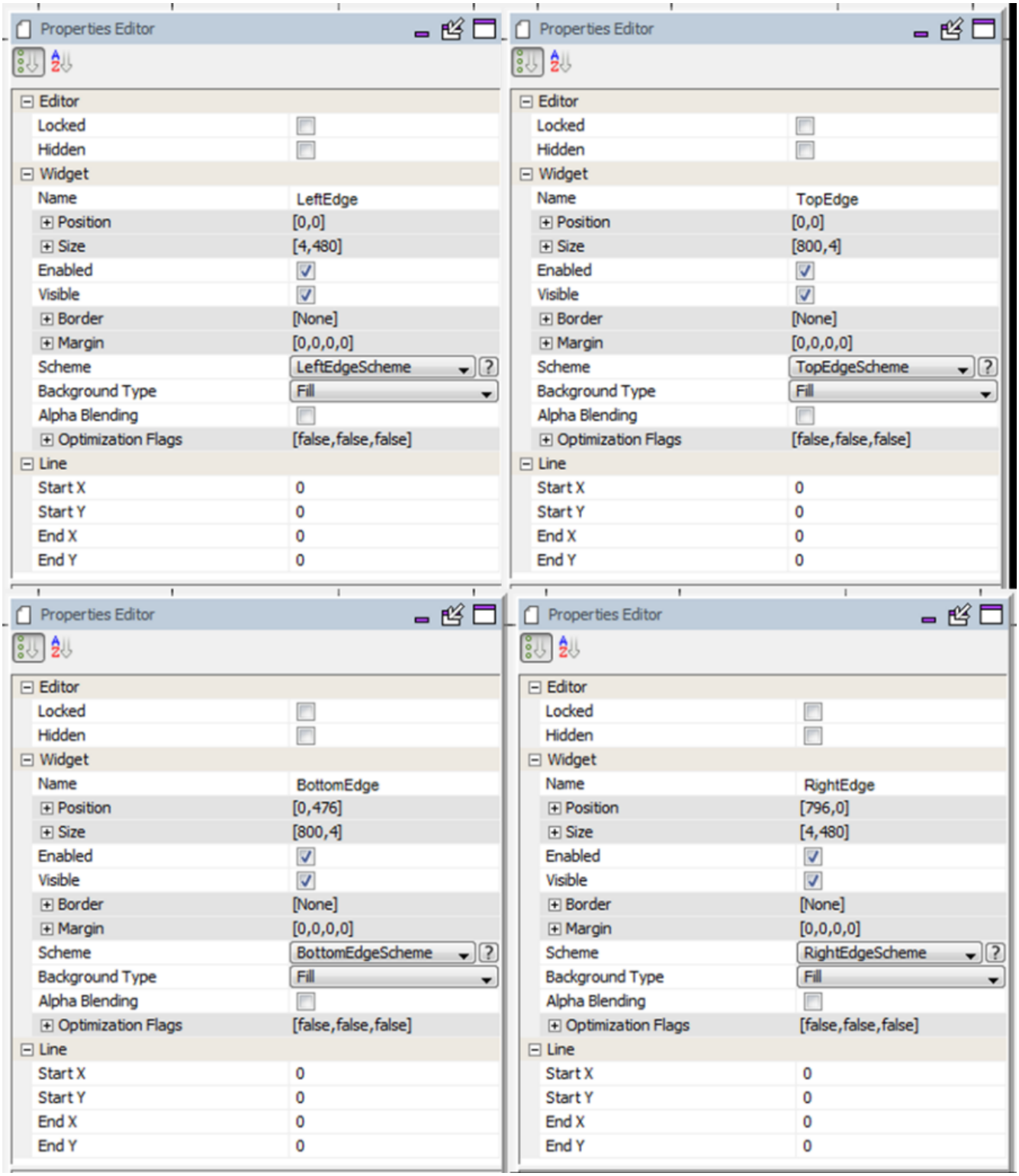


Using the Scheme panel, create four new color schemes.

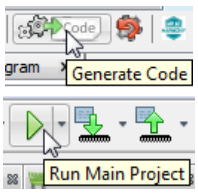




Next, drag a line widget onto the display four times and edit each widget's properties to create and position each edge of the display's border:



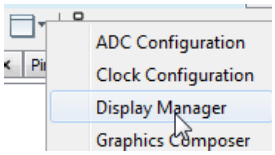
Note that the “Line” coordinates are set to [0,0,0,0] since it is the size of the widget rather than the widget’s line that marks each border line. The lines in these widgets are not used. Each widget’s position and size mark an edge of the display, not the line. Re-generate the application and then run it.



The HyperTerminal application (115200 baud, 8 bits, no stop bits) should show the following when the application boots up:

```
SSD1963 driver LCDC_FPR: 314574, (0x04CCCE)

Application created Mar  4 2018 17:51:00 initialized!
```

Examine the border of the resulting display,note that the top edge of the border is completely missing and the left edge is about half the width desired, compared to the right and bottom edges. To fix this the display timings need to be adjusted using the Display Manager:

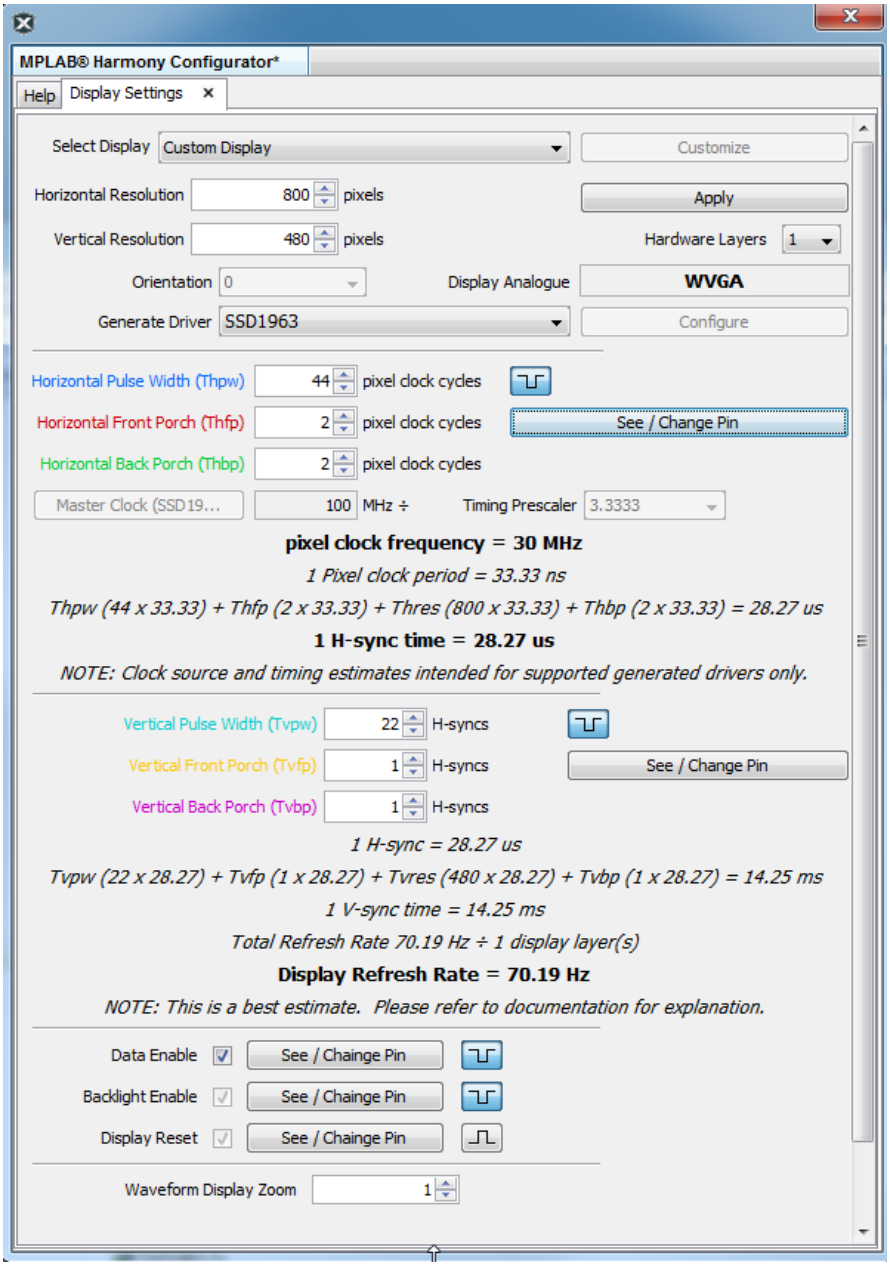


If this is the first time using the Display Manger, Volume 1 of MPLAB Harmony's documentation has a Display Manager Quick Start Guide and Volume III has the MPLAB Harmony Display Manager User's Guide. Increase the Horizonal Pulse Width by two clocks, re-generate, and then run. The left border should be fully visible.

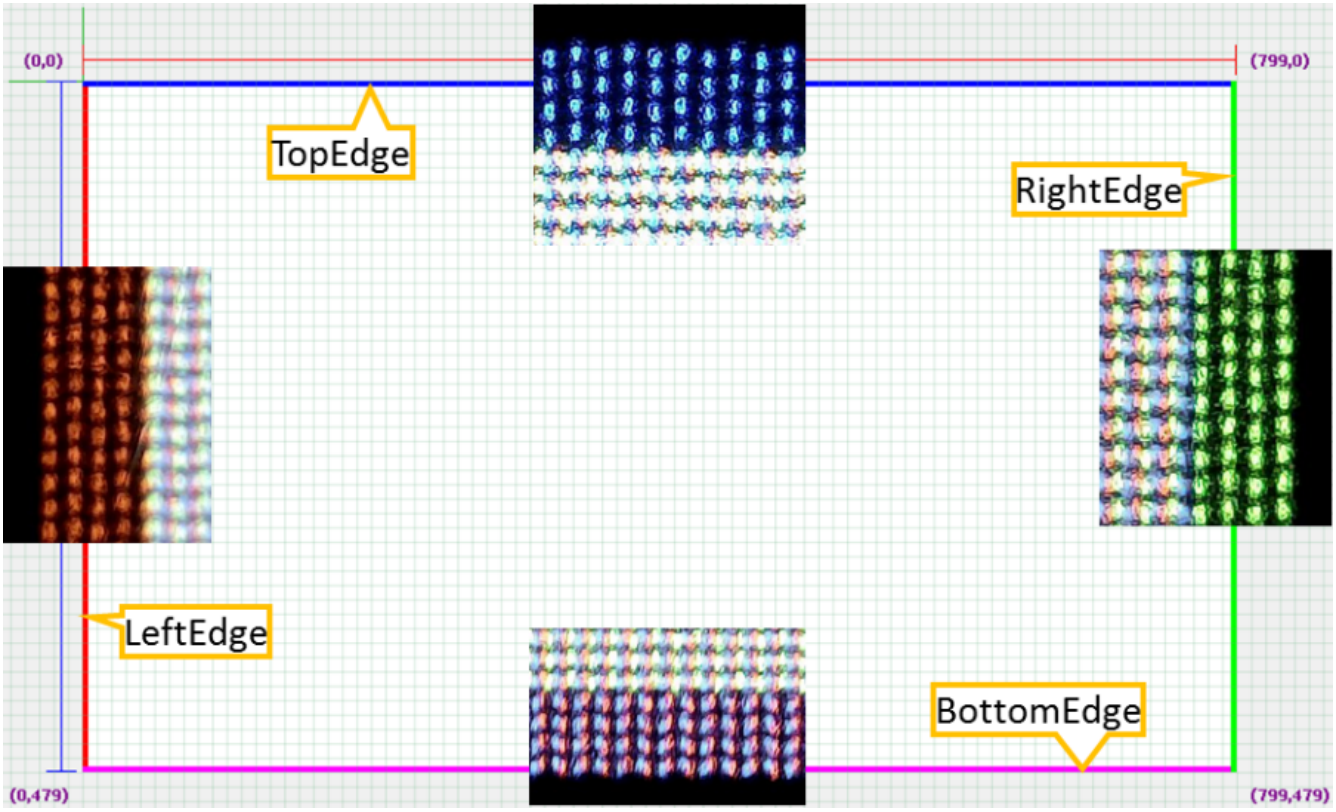
Next, tune the Vertical Pulse Width. Gradually increasing it to move the top border line down until it is fully visible. (22 H-syncs seems to be the correct value.)

After each adjustment re-generate, build and run, then examine the resulting display. Stop when all borders are fully visible and there are no "dead" (black) pixels on the display.

In the Display Manger, the final, optimal, settings for the display are:



When finished, the display should be:



A picture of each edge through a 10x power loupe verifies that each edge is exactly 4 pixels wide and there are no “dead” (black) pixels between the edges of the display and the colored border.

The Mikroe board uses a Riverdi RVT50AQTNWC00 display.

Table 8.3 of its datasheet covers display timing:

8.3 Parallel RGB timing table

PARAMETER	SYMBOL	MIN	TYP	MAX	UNIT
Horizontal Display Area	Thd	-	800	-	DCLK
DCLK Frequency	Fclk	-	30	50	MHz
One Horizontal Line	Th	889	928	1143	DCLK
HS pulse width	Thpw	1	48	255	DCLK
HS Blanking	Thb	-	88	-	DCLK
HS Front Porch	Thfp	1	40	255	DCLK
Vertical Display Area	Tvd	-	480	-	TH
VS period time	Tv	513	525	767	TH
VS pulse width	Tvpw	3	3	255	TH
VS Blanking	Tvb	-	32	-	TH
VS Front Porch	Tvfp	1	13	255	TH

Some explanation is required to match up this data with the Display Manager’s settings. Back porch timings are not shown in the table, but can be calculated by subtracting the HS/VS pulse width from the HS/VS Blanking:

HS Back Porch = HS Blanking – HS pulse width = Thbp = Thb – Thfp

VS Back Porch = VS Blanking – VS pulse width = Tvbp = Tvb – Tvfp

The DCLK Frequency typical value of 30 MHz has already been used in setting up the display pixel clock speed. However, using the “Typ” (Typical) values, and the calculated Thbp and Tvbp values from the equations above, the timing will not work. The timing values that work for this tutorial meet the minimum or maximum range shown above with one exception:

The “One Horizontal Line” timing, Th, has a minimum of 889 pixel clocks, but the one in use is:

Th = Thpw + Thbp + 800 pixels+ Thfp = 44 + 2 +800 + 2 = 848 pixel clocks < 889

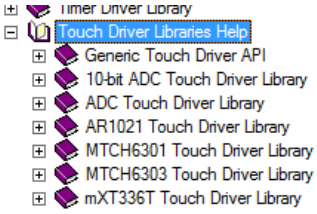
which is 41 pixel clocks (4.6%) below the minimum Th of 889 shown in Table 8.3 above.

Results may vary on your display. This was tested on two different boards with the same results. Starting out with the default display timings and then iteratively tuning them to reduce pixel clipping and dead pixels, as discussed above, will provide the optimal display timings for the hardware regardless of the final settings.

Supporting the Focal Tech FT5x06 Capacitive Touch Controller

Microchip (Atmel) and Focal Tech are key providers of capacitive touch controllers. Focal Tech FT5x06 touch controllers are found on many of the displays used by Microchip customers, so a third-party display with a Focal Tech capacitive touch controller is a good choice for this tutorial.

MPLAB Harmony provides these touch controller drivers:



The Generic Touch Driver outlines the generic Touch Driver API supported by MPLAB Harmony. It provides a template that can serve as the base for a custom-built driver for the FT5x06 touch controller.

A faster way to support the Focal Tech FT5x06 is to find a similar device that is already supported in MPLAB Harmony and simply modify the driver code for that device. This eliminates having to write all the supporting code needed to fit the new driver into MPLAB Harmony. Capacitive touch devices typically have an I2C interface with the host, and an interrupt signal that is driven low to alert the host that a touch event has been detected. In response to this external interrupt the host uses the I2C interface with the device to query the device and read the (x,y) pixel coordinates of the touch event.

The FT5x06 command interface is closest to the MTCH6303 interface since it requires a write command followed by a read command to get the touch event. (The MTCH6301 only requires the read message.) The other thing to be aware of is the data order coming from the chip.

FT5x06 Memory:

Operating Mode Register Map

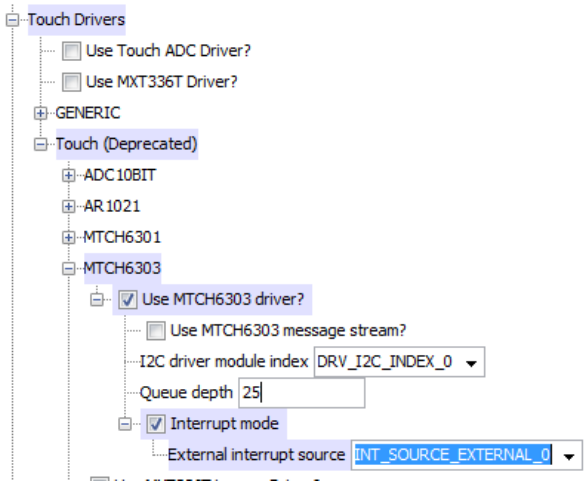
Address	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Host Access
Op,00h	DEVIDE_MODE		Device Mode[2:0]							RW
Op,01h	GEST_ID	Gesture ID[7:0]								R
Op,02h	TD_STATUS					Number of touch points[3:0]				R
Op,03h	TOUCH1_XH	1st Event Flag				1st Touch X Position[11:8]				R
Op,04h	TOUCH1_XL	1st Touch X Position[7:0]								R
Op,05h	TOUCH1_YH	1st Touch ID[3:0]				1st Touch Y Position[11:8]				R
Op,06h	TOUCH1_YL	1st Touch Y Position[7:0]								R
Op,07h										
Op,08h										
Op,09h	TOUCH2_XH	2nd Event				2nd Touch				R

		Flag		X Position[11:8]	
Op,0Ah	TOUCH2_XL	2nd touch X Position[7:0]			R
Op,0Bh	TOUCH2_YH	2nd Touch ID[3:0]		2nd Touch Y Position[11:8]	R
Op,0Ch	TOUCH2_YL	2nd Touch Y Position[7:0]			R
Op,0Dh					R
Op,0Eh					R
Op,0Fh	TOUCH3_XH	3rdEvent Flag		3rd Touch X Position[11:8]	R
Op,10h	TOUCH3_XL	3rd Touch X Position[7:0]			R
Op,11h	TOUCH3_YH	3rd Touch ID[3:0]		3rd Touch Y Position[11:8]	R
Op,12h	TOUCH3_YL	3rd Touch Y Position[7:0]			R
Op,13h					R
Op,14h					R
Op,15h	TOUCH4_XH	4thEvent Flag		4th Touch X Position[11:8]	R
Op,16h	TOUCH4_XL	4th Touch X Position[7:0]			R
Op,17h	TOUCH4_YH	4th Touch ID[3:0]		4th Touch Y Position[11:8]	R
Op,18h	TOUCH4_YL	4th Touch Y Position[7:0]			R
Op,19h					R
Op,1Ah					R
Op,1Bh	TOUCH5_XH	5thEvent Flag		5th Touch X Position[11:8]	R
Op,1Ch	TOUCH5_XL	5thTouch X Position[7:0]			R
Op,1Dh	TOUCH5_YH	5th Touch ID[3:0]		5th Touch Y Position[11:8]	R
Op,1Eh	TOUCH5_YL	5th Touch Y Position[7:0]			R
Op,1Fh					R
Op,20h					R
…	…				
Op,7Fh	Reserved				
Op,80h	ID_G_THGROUP	valid touching detect threshold.			R/W
Op,81h	ID_G_THPEAK	valid touching peak detect threshold.			R/W

Modifying MPLAB Harmony's MTCH6303 Touch Driver for the Focal Tech FT5x06

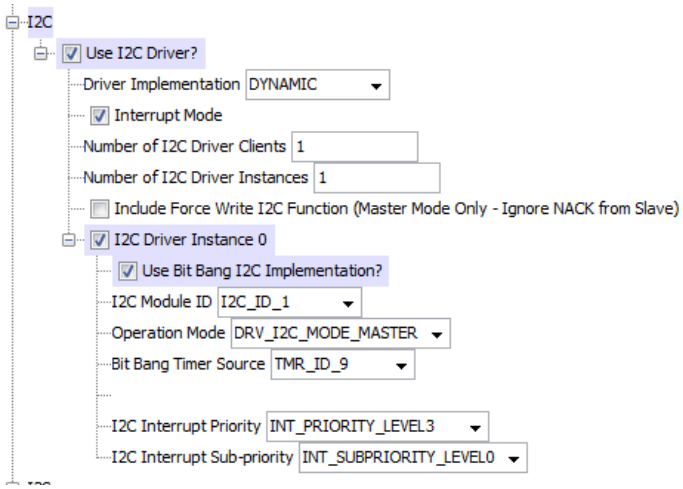
The first step towards supporting the FT5x06 is to add a MTCH6303 driver to the application, and then modify the MTCH6303's code to support the FT5x06. To support the FT5x06, we will add a C preprocessor #if defined(FT_SUPPORT)…#else…#endif clauses to the code and then define FT_SUPPORT in the project's C compiler properties.

To add the MTCH6303 touch driver, make the following changes to the project's MHC settings:

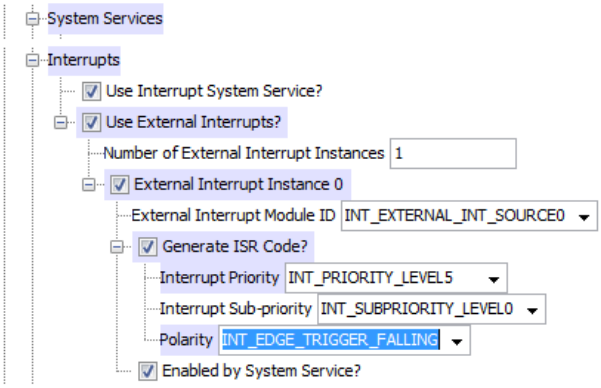


Be sure to increase the event queue depth from the default of 10 to something larger, here it is 25. The controller's CTP-INT# (CTP_INT_BAR in the Pin Settings table) is connected to INT0, so change the external interrupt source to INT_SOURCE_EXTERNAL_0.

Next, enable the I2C driver, using a bit-banged implementation:



The Interrupt System Service is enabled, with an Interrupt Priority of 5, connected to INT0, and triggered on a falling edge (since CTP-INT# is active low):



Re-generate the application to implement these changes to the application.

Rather than edit the application's MTCH6303 driver code, install the modified driver from the tutorial project found in `.\apps\examples\3rdPartyDisplay`. Copy the code found in directory

`.\apps\examples\3rdPartyDisplay\firmware\src\system_config\default\framework\driver\touch\mtch6303`

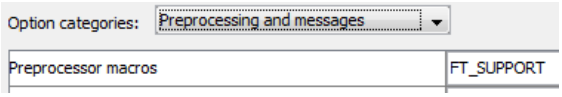
into the same folder in the project.

To keep these changes in the code whenever the project is regenerated, always choose the "Prompt Merge For All Differences" merge strategy and simply close all the windows related to the MTCH6303 driver. These changes are identified by // CUSTOM CODE – DO NOT DELETE … // END OF CUSTOM CODE flags in the code.

Note: Ignore all proposed changes for the following files:

- `drv_mtch6303_static.h`
- `drv_mtch6303_static.c`
- `drv_mtch6303_static_local.h`

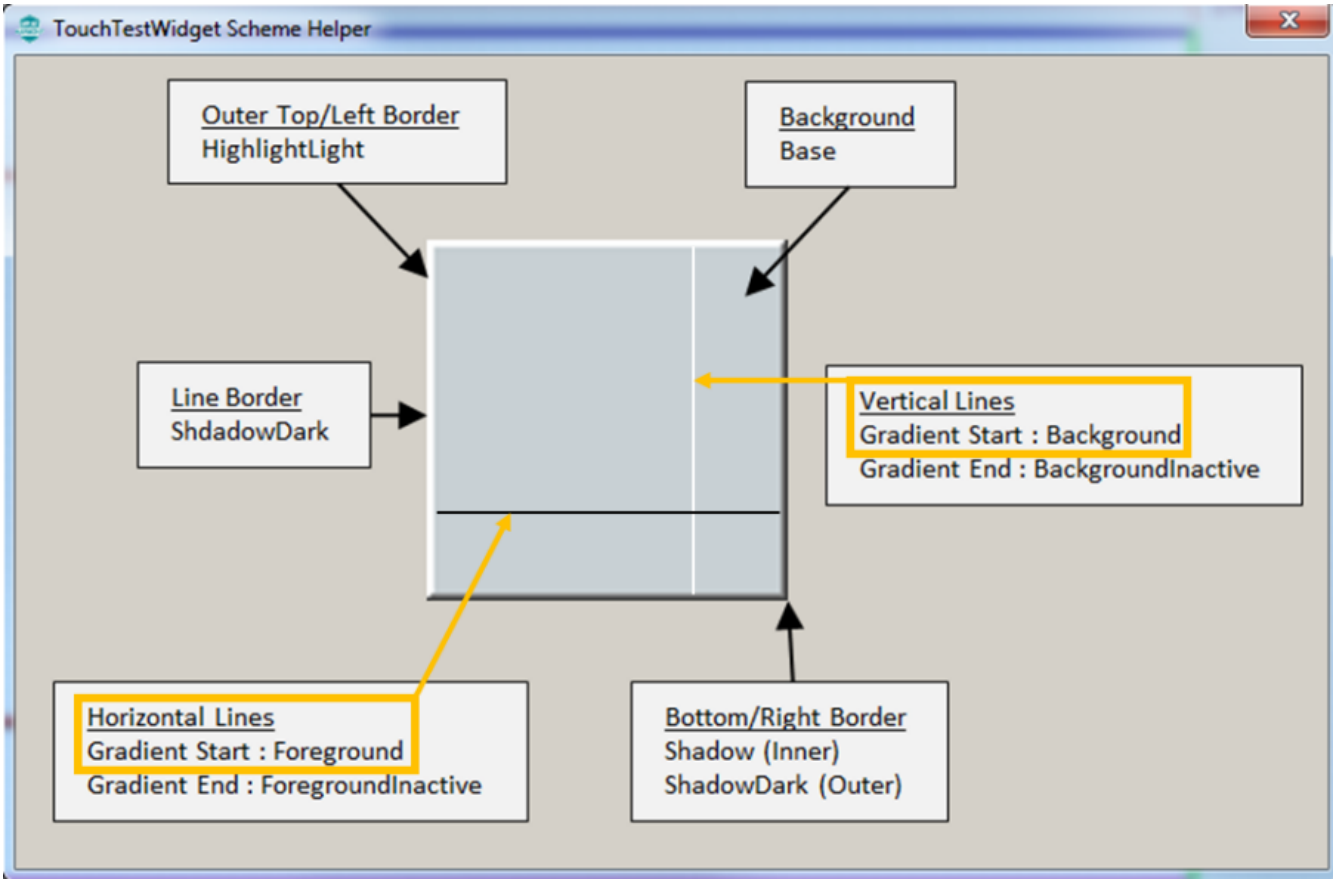
To enable Focal Tech support in the modified driver, open the project's configuration and define FT_SUPPORT in the C compiler section.



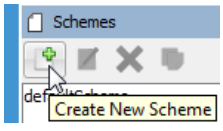
Adding a Touch Test Widget

Bring up MHC's Graphics Composer again and add a Touch Test widget to the screen. Resize the widget to cover most of the display. Next, create another color scheme, and customize it to see the cross hairs for all touch measurements reported by the widget.

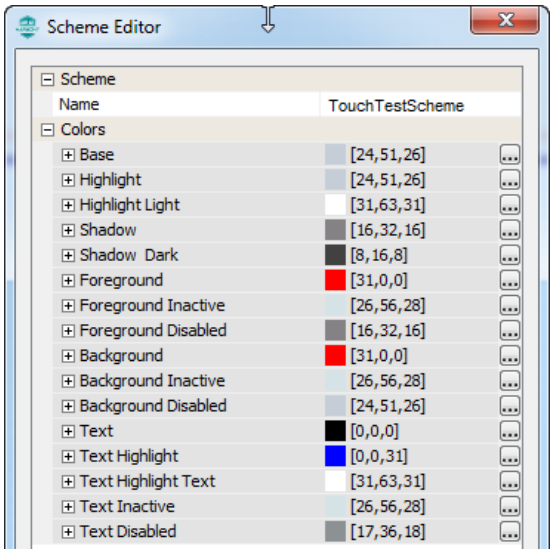
The TouchTest Widget has the following color scheme:



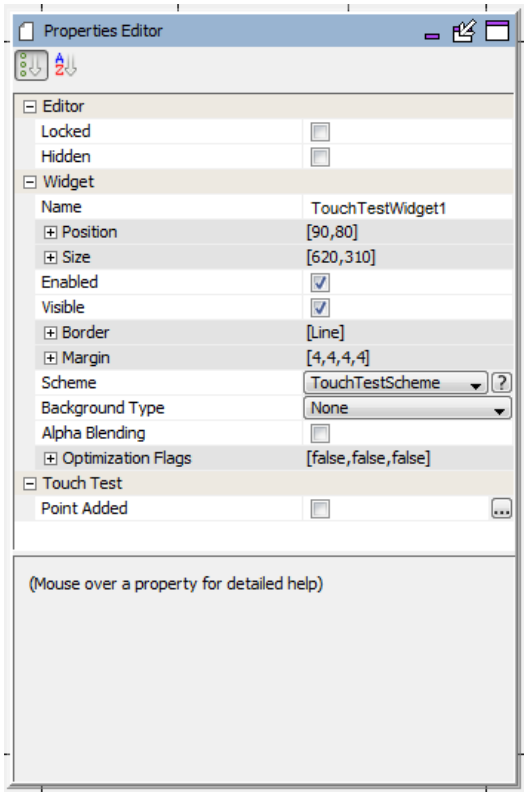
First, create a new scheme, call it TouchTestScheme:



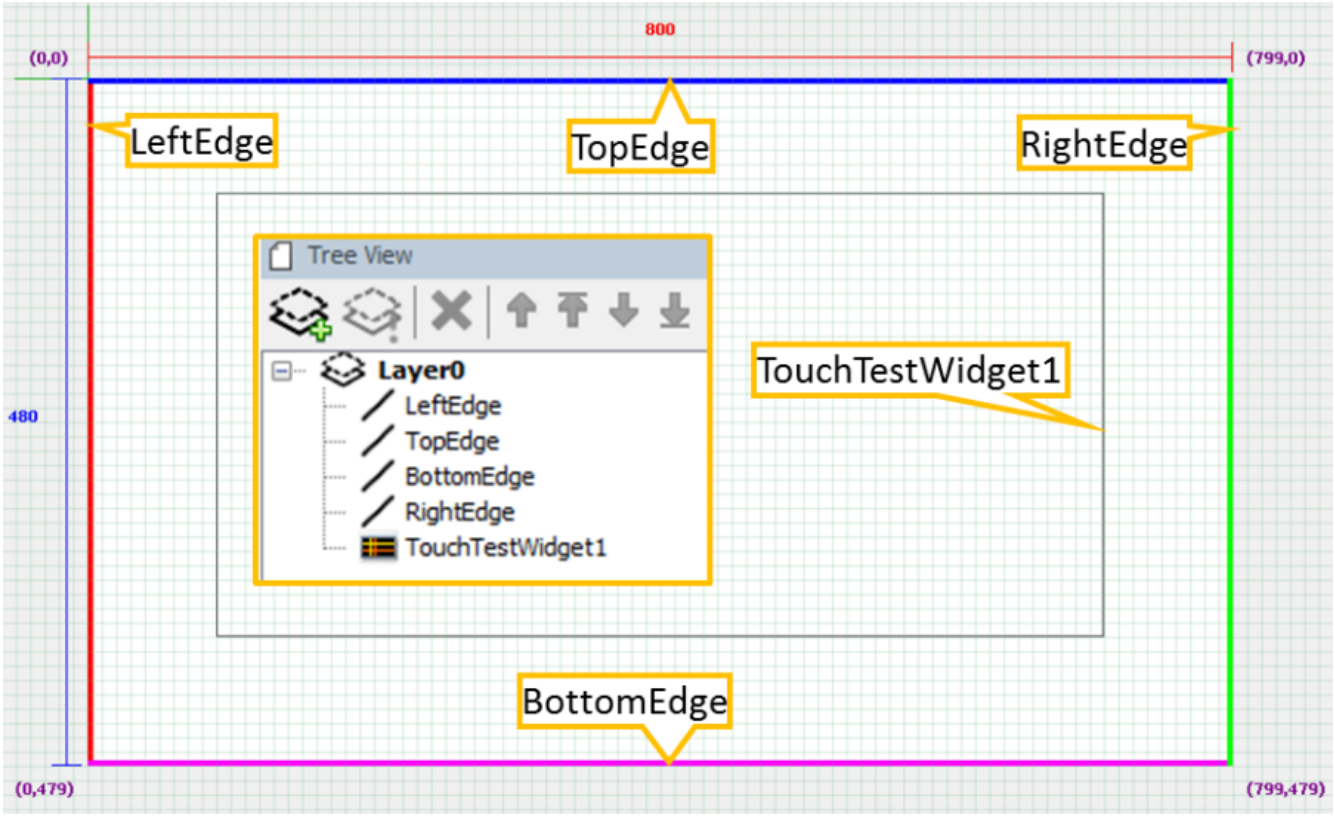
Edit the Foreground and Background colors so that both are red.



Finally, edit the properties for the Touch Test widget to have a Line border, and to use the TouchTestScheme color scheme:



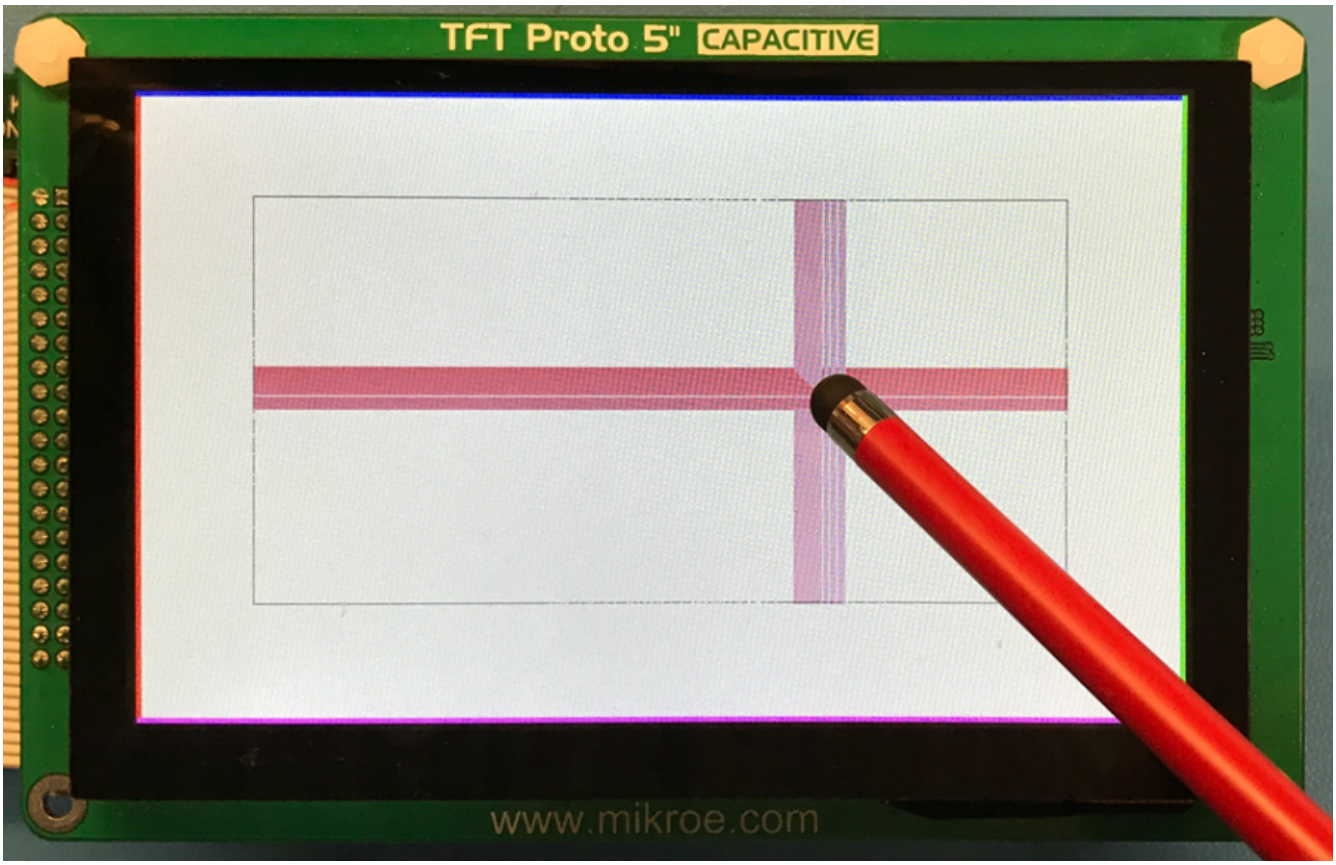
The Screen Designer panel should show:



Close the Graphics Composer window and save the modifications to the graphics design. Re-generate the application's code and then build and load the application.

Testing the Final Application

Here is what the display should look like during a touch event:



Completed Tutorial Project

The completed tutorial project can be found in .\apps\examples\3rdPartyDisplay.

Importing and Exporting Graphics Data

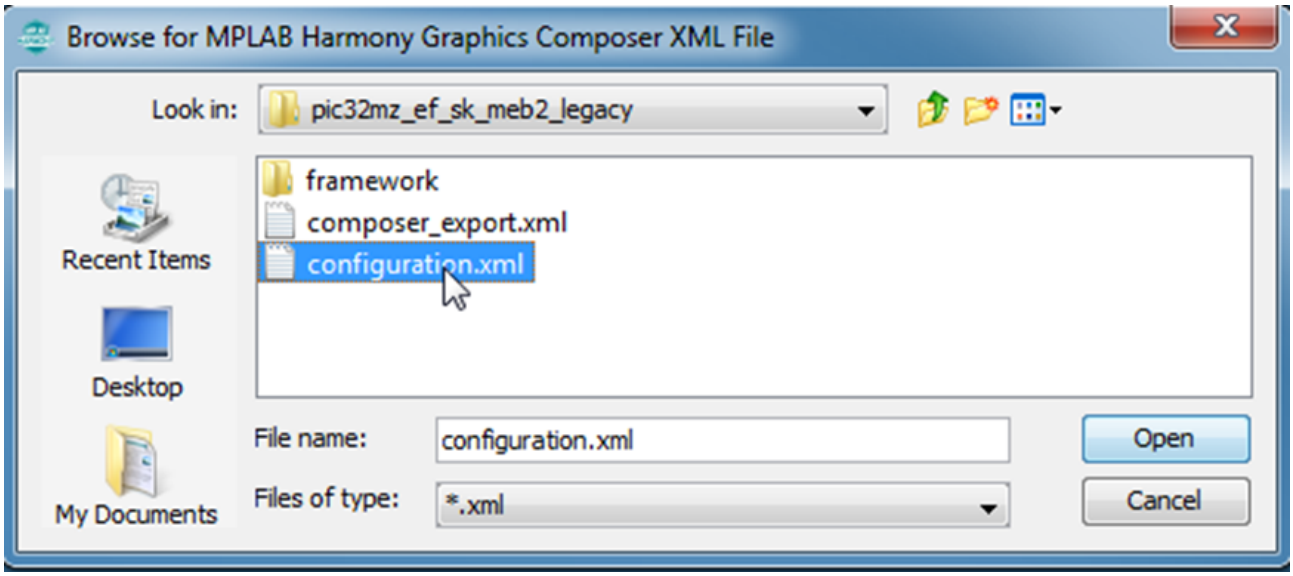
This topic provides information on importing and exporting graphics composer-related data.

Description

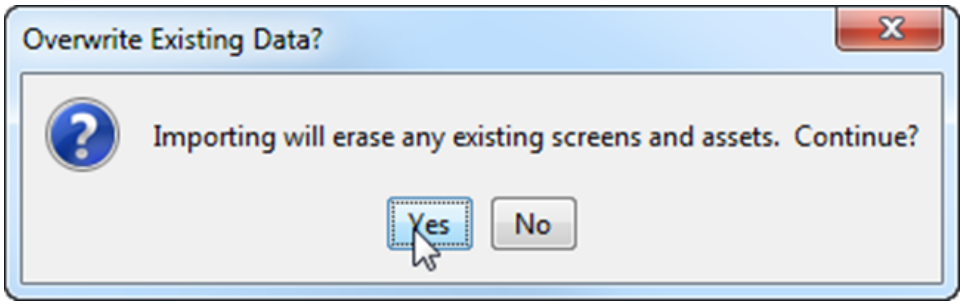
The MPLAB Harmony Graphics Composer (MHGC) provides the capability for users to import and export graphics designs. The user can export the state of an existing graphics composer configuration or import another graphics composer configuration from another project.

Importing Data

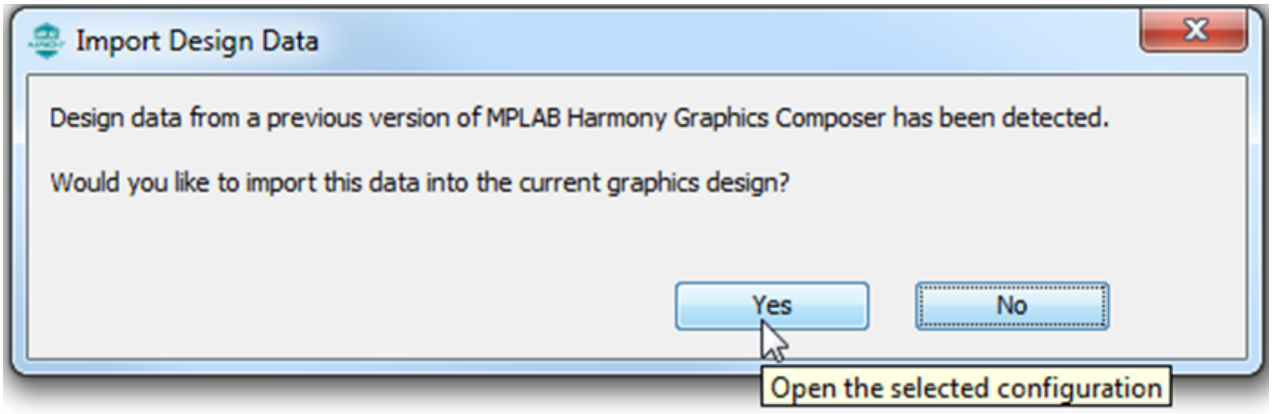
1. To import a graphics design into MHGC, select *File* > *Import*. The Browse for MPLAB Harmony Graphics Composer XML file dialog appears, which allows the selection of a previously exported Graphics Composer `.xml` file, or the `configuration.xml` file that contains the desired graphics image.



2. After selecting a file and clicking **Open**, you will be prompted whether to overwrite existing data.

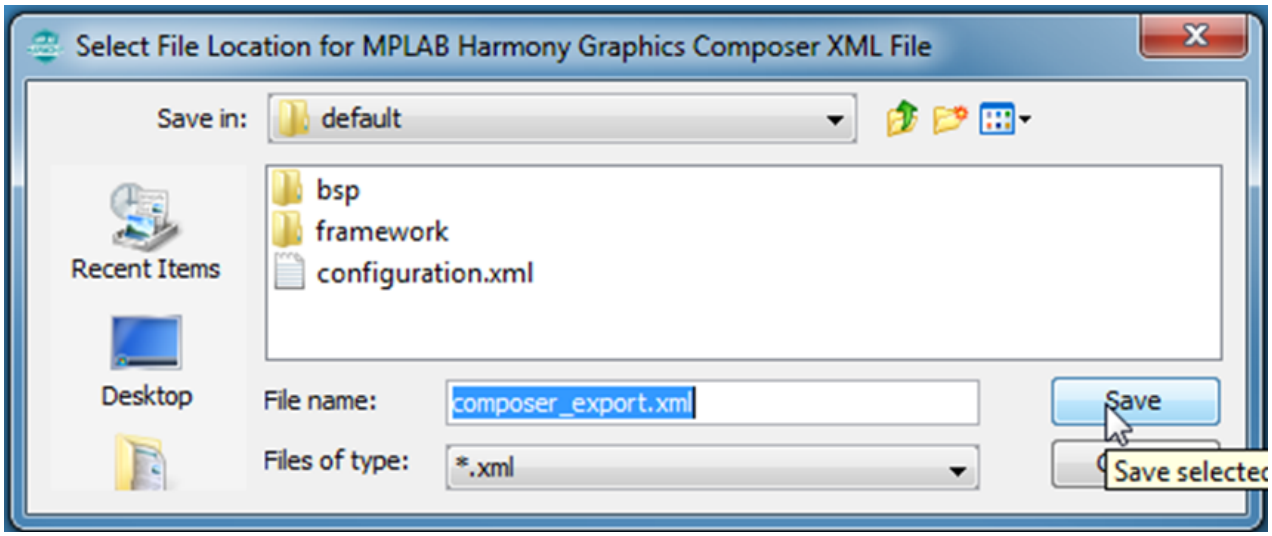


3. If you selected a `composer_export.xml` file, clicking **Yes** will replace the current graphics design with the new design.
4. Otherwise, if you selected a `configuration.xml` file, you will be prompted to import the data into the current graphics design. Click **Yes** to replace the current graphics design with the new design.

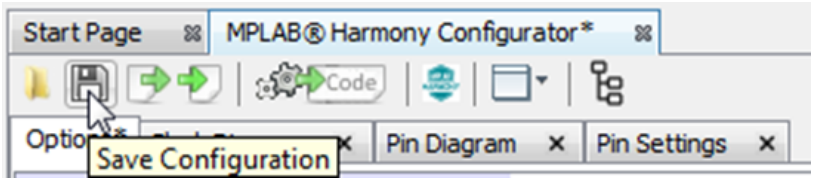


Exporting Data

1. To export a graphics design from MHGC, select *File* > *Export*. The Select File Location for MPLAB Harmony Graphics Composer XML file dialog appears.



2. To export a graphics design using a `configuration.xml` file, use the **Save Configuration** utility from the MPLAB Harmony Configurator (MHC) toolbar.



MPLAB Harmony Graphics Composer Suite

This section provides user information about using the MPLAB Harmony Graphics Composer Suite (MHGS).

Description

Please see *Volume IV: MPLAB Harmony Framework Reference > Graphics Libraries Help > MPLAB Harmony Graphics Composer Suite* for detailed information.

Index

P

S

T

U

V

W